

Jianzhong Wang

Geometric Structure of High-Dimensional Data and Dimensionality Reduction



高等教育出版社
HIGHER EDUCATION PRESS



Springer

Jianzhong Wang

Geometric Structure of High-Dimensional Data and Dimensionality Reduction

Jianzhong Wang

Geometric Structure of High-Dimensional Data and Dimensionality Reduction

With 91 figures



Author

Prof. Jianzhong Wang
Department of Mathematics and Statistics
Sam Houston State University
1901 Avenue I, Huntsville
TX 77382-2206, USA
E-mail: jzwang@shsu.edu

ISBN 978-7-04-031704-6
Higher Education Press, Beijing

ISBN 978-3-642-27496-1 ISBN 978-3-642-27497-8 (eBook)
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011944174

© Higher Education Press, Beijing and Springer-Verlag Berlin Heidelberg 2012

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To my wife Huilan and my sons Chunhui and Chunze

Preface

Many objects in our world can be electronically represented with high-dimensional data—speech signals, images, videos, electrical text documents. We often need to analyze a large amount of data and process them. However, due to the high dimension of these data, directly processing them using regular systems may be too complicated and unstable to be feasible. In order to process high-dimensional data, dimensionality reduction technique becomes crucial. Dimensionality reduction is a method to represent high-dimensional data by their low-dimensional embeddings so that the low-dimensional data can be effectively used either in processing systems, or for better understanding. This technique has proved an important tool and has been widely used in many fields of data analysis, data mining, data visualization, and machine learning.

Principal component analysis (PCA) and classical multidimensional scaling (CMDS) are two traditional methods based on linear data models. However, they are not effective if data do not well reside on superplanes. Recently, many methods of nonlinear dimensionality reduction, also called manifold learning, have been developed based on nonlinear data models, which assume each observed high-dimensional data resides on a low-dimensional manifold. A nonlinear method *learns* the underlying manifold from the data and then represents the data by its low-dimensional manifold coordinates.

This monograph is an introductory treatise on dimensionality deduction with an emphasis on nonlinear methods. It includes a comprehensive introduction to linear methods, and then focuses on the geometric approach to nonlinear dimensionality reduction. The approach adopts manifold embedding technique that embeds a high-dimensional dataset into a low-dimensional manifold. The main idea of this approach can be outlined as follows. The geometry of the underlying manifold is learned from the neighborhood structure of the data, which defines the data similarity. Then a kernel is created to represent the learned manifold geometry and the spectral decomposition of the kernel leads to the manifold embedding. The different nonlinear methods adopt different manifold learning models. I hope that this book will illustrate this idea effectively.

Giving consideration to various readers, I try to keep a balance between

intuitive description and strictly mathematical deduction in this book. Each method is first described with the intuitive idea. Then the formulation is derived with necessary details. Finally, the algorithm is presented and interpreted. Graphs are used to illustrate the capability and validity, as well as the advantages and shortcomings, of each method. Several applications, such as face recognition, document ordering and classification, objective visualization, are introduced or outlined. I have no ambition to completely cover these applications. I hope the introduction to them will help readers understand the concepts and contents of dimensionality reduction. The technical mathematical justification is put in the last section of each chapter. The readers who are only interested in the applications of dimensionality reduction can skip, for example, Chapter 2, Section 3.3, and all justification sections.

This monograph first presents a general introduction. It then presents the material in three parts. In Part I, the geometry of data is discussed. In this part, there is a brief review of manifold calculus, followed by a preliminary presentation of spectral graph theory. These are the main tools in the study of the geometry of data. The last chapter of Part I discusses the data models and strategies of dimensionality reduction. In Part II, linear methods are discussed. Besides PCA and CMDS, the random projection methods are also introduced. The main part of this monograph is Part III, in which several popular nonlinear methods are introduced, such as Isomaps, maximum variance unfolding (MVU), locally linear embedding (LLE), local tangent space alignment (LTSA), Laplacian eigenmaps (Leigs), Hessian locally linear embedding (HLLE), and diffusion maps (Dmaps). This part ends with a chapter on fast dimensionality reduction algorithms, which have attracted more interest recently, with the increasing availability of very large data sets.

The book is primarily intended for computer scientists, applied mathematician, statisticians, and data analysts. It is also accessible to other audiences who want to apply dimensionality reduction technique to understanding and treating the complex data in their own fields, such as economists and geophysics.

I am very grateful to my colleague and friend Dr. Barry Friedman who has read the manuscript. To the editorial office, particularly to Dr. Ying Liu, I wish to express my appreciation of their efficient assistance and friendly cooperation.

Jianzhong Wang
Huntsville, Texas
February, 2011

Contents

Chapter 1 Introduction	1
1.1 Overview of Dimensionality Reduction	1
1.2 High Dimension Data Acquisition	3
1.2.1 Collection of Images in Face Recognition	3
1.2.2 Handwriting Letters and Digits	5
1.2.3 Text Documents	6
1.2.4 Hyperspectral Images	6
1.3 Curse of the Dimensionality	8
1.3.1 Volume of Cubes and Spheres	9
1.3.2 Volume of a Thin Spherical Shell	9
1.3.3 Tail Probability of the Multivariate Gaussian Distributions	10
1.3.4 Diagonals of Cube	11
1.3.5 Concentration of Norms and Distances	11
1.4 Intrinsic and Extrinsic Dimensions	12
1.4.1 Intrinsic Dimension Estimation	12
1.4.2 Correlation Dimension	15
1.4.3 Capacity Dimension	16
1.4.4 Multiscale Estimation	16
1.5 Outline of the Book	17
1.5.1 Categories of DR Problems	17
1.5.2 Scope of This Book	18
1.5.3 Other Topics Related to This Book	21
1.5.4 Artificial Surfaces for Testing DR Algorithms	22
References	24

Part I Data Geometry

Chapter 2 Preliminary Calculus on Manifolds	29
2.1 Linear Manifold	29
2.1.1 Subspace and Projection	30
2.1.2 Functions on Euclidean Spaces	32
2.1.3 Laplace Operator and Heat Diffusion Kernel	34
2.2 Differentiable Manifolds	35
2.2.1 Coordinate Systems and Parameterization	35
2.2.2 Tangent Spaces and Tangent Vectors	38
2.2.3 Riemannian Metrics	43
2.2.4 Geodesic Distance	45
2.3 Functions and Operators on Manifolds	46
2.3.1 Functions on Manifolds	46
2.3.2 Operators on Manifolds	47
References	49
 Chapter 3 Geometric Structure of High-Dimensional Data	 51
3.1 Similarity and Dissimilarity of Data	51
3.1.1 Neighborhood Definition	52
3.1.2 Algorithms for Construction of Neighborhood	56
3.2 Graphs on Data Sets	60
3.2.1 Undirected Graphs	60
3.2.2 Directed Graphs	64
3.2.3 Neighborhood and Data Graphs	66
3.3 Spectral Analysis of Graphs	67
3.3.1 Laplacian of Graphs	67
3.3.2 Laplacian on Weighted Graphs	74
3.3.3 Contracting Operator on Weighted Graph	75
References	76
 Chapter 4 Data Models and Structures of Kernels of DR	 79
4.1 Data Models in Dimensionality Reduction	79
4.1.1 Input Data of First Type	79
4.1.2 Input Data of Second Type	81
4.1.3 Constraints on Output Data	82
4.1.4 Consistence of Data Graph	83
4.1.5 Robust Graph Connection Algorithm	84
4.2 Constructions of DR Kernels	87
4.2.1 DR Kernels of Linear Methods	87

4.2.2 DR Kernels of Nonlinear Methods	88
4.2.3 Conclusion	90
References	91

Part II Linear Dimensionality Reduction

Chapter 5 Principal Component Analysis	95
5.1 Description of Principal Component Analysis	95
5.1.1 Geometric Description of PCA	95
5.1.2 Statistical Description of PCA	99
5.2 PCA Algorithms	100
5.2.1 Matlab Code for PCA Algorithm	101
5.2.2 EM-PCA Algorithm	102
5.2.3 Testing PCA Algorithm on Artificial Surfaces	104
5.3 Applications of PCA	106
5.3.1 PCA in Machine Learning	107
5.3.2 PCA in Eigenfaces	108
5.3.3 PCA in Hyperspectral Image Analysis	112
References	113
Chapter 6 Classical Multidimensional Scaling	115
6.1 Introduction of Multidimensional Scaling	115
6.1.1 Data Similarities and Configuration	115
6.1.2 Classification of MDS	117
6.2 Euclidean Matric and Gram Matrices	118
6.2.1 Euclidean Distance Matrices	119
6.2.2 Gram Matrix on Data Set	120
6.2.3 Relation between Euclidean Distance and Gram Matrix	120
6.3 Description of Classical Multidimensional Scaling	123
6.3.1 CMDS Method Description	123
6.3.2 Relation between PCA and CMDS	126
6.3.3 Weighted Graphic Description of CMDS	126
6.4 CMDS Algorithm	126
References	128
Chapter 7 Random Projection	131
7.1 Introduction	131
7.1.1 Lipschitz Embeddings	131
7.1.2 JL-Embeddings	132

7.2	Random Projection Algorithms	133
7.2.1	Random Matrices and Random Projection	133
7.2.2	Random Projection Algorithms	135
7.3	Justification	136
7.3.1	Johnson and Lindenstrauss Lemma	136
7.3.2	Random Projection based on Gaussian Distribution	138
7.3.3	Random Projection based on Types 2 and 3	141
7.4	Applications of Random Projections	144
7.4.1	Face Recognition Experiments with Random Projection	145
7.4.2	RP Applications to Image and Text Data	146
	References	147

Part III Nonlinear Dimensionality Reduction

Chapter 8 Isomaps	151	
8.1	Isomap Embeddings	151
8.1.1	Description of Isomaps	151
8.1.2	Geodesic Metric on Discrete Set	153
8.1.3	Isomap Kernel and its Constant Shift	153
8.2	Isomap Algorithm	156
8.2.1	Algorithm Description	156
8.2.2	Matlab Code of Isomap	158
8.3	Dijkstra's Algorithm	160
8.3.1	Description of Dijkstra's Algorithm	160
8.3.2	Matlab Code of Dijkstra's Algorithm	163
8.4	Experiments and Applications of Isomaps	169
8.4.1	Testing Isomap Algorithm on Artificial Surfaces	169
8.4.2	Isomap Algorithm in Visual Perception	172
8.4.3	Conclusion	174
8.5	Justification of Isomap Methods	175
8.5.1	Graph Distance, S-distance, and Geodesic Distance	175
8.5.2	Relation between S-distance and Geodesic Distance	176
8.5.3	Relation between S-distance and Graph Distance	177
8.5.4	Main Result	178
	References	179
Chapter 9 Maximum Variance Unfolding	181	
9.1	MVU Method Description	181
9.1.1	Description of the MVU Method	183

9.1.2	MVU Algorithm	185
9.2	Semidefiniteness Programming	186
9.2.1	CSDP	187
9.2.2	SDPT3	188
9.3	Experiments and Applications of MVU	188
9.3.1	Testing MVU Algorithm on Artificial Surfaces	188
9.3.2	MVU Algorithm in Sensor Localization	190
9.4	Landmark MVU	194
9.4.1	Description of Landmark MVU	194
9.4.2	Linear Transformation from Landmarks to Data Set	195
9.4.3	Algorithm for Landmark Linear Transformation	198
9.4.4	Construction of Kernel of Landmark MVU	199
9.4.5	Experiments of LMVU	200
9.4.6	Conclusion	200
	References	201
Chapter 10 Locally Linear Embedding		203
10.1	Description of Locally Linear Embedding	203
10.1.1	Barycentric Coordinates	204
10.1.2	LLE Method	205
10.1.3	LLE Algorithm	207
10.2	Experiments and Applications of LLE	210
10.2.1	Experiments on Artificial Surfaces	210
10.2.2	Conclusion	213
10.3	Applications of LLE	214
10.3.1	LLE in Image Ordering	214
10.3.2	Supervised LLE	215
10.4	Justification of LLE	216
10.4.1	Invariance Constraint	216
10.4.2	Condition for Weight Uniqueness	218
10.4.3	Reduction of the DR Data to LLE Model	219
	References	220
Chapter 11 Local Tangent Space Alignment		221
11.1	Description of Local Tangent Space Alignment	221
11.1.1	Tangent Coordinates and Manifold Coordinates	221
11.1.2	Local Coordinate Representation	223
11.1.3	Global Alignment	225
11.2	LTSA Algorithm	226

11.2.1	LTSA Algorithm Description	226
11.2.2	Matlab Code of LTSA	227
11.3	Experiments of LTSA Algorithm	229
11.3.1	Test LTSA on Artificial Surfaces	229
11.3.2	Conclusion	233
References		233
Chapter 12	Laplacian Eigenmaps	235
12.1	Description of the Laplacian Eigenmap Method	235
12.1.1	Approximation of Laplace-Beltrami Operator	236
12.1.2	Discrete form of Laplace-Beltrami Operator	237
12.1.3	Minimization Model for DR Data Set	238
12.1.4	Construction of General Leigs Kernels	239
12.2	Laplacian Eigenmaps Algorithm	240
12.2.1	Steps in Leigs Algorithm	240
12.2.2	Matlab Code of Leigs Algorithm	241
12.3	Implementation of Leigs Algorithm	243
12.3.1	Experiments on Artificial Surfaces	243
12.3.2	Conclusion	247
References		247
Chapter 13	Hessian Locally Linear Embedding	249
13.1	Description of Hessian Locally Linear Embedding	249
13.1.1	Hessian on Manifold	250
13.1.2	Hessian on Tangent Space	251
13.1.3	Construction of Hessian Functional	252
13.1.4	Construct of HLLE DR Kernel	255
13.2	HLLE Algorithm	256
13.2.1	HLLE Algorithm Description	256
13.2.2	Matlab Code of HLLE	257
13.3	Implementation of HLLE	260
13.3.1	Experiments on Artificial Surfaces	260
13.3.2	Conclusion	265
References		265
Chapter 14	Diffusion Maps	267
14.1	Description of DR Method of Diffusion Maps	267
14.1.1	Diffusion Operator on Manifold	268
14.1.2	Normalization of Diffusion Kernels	269
14.2	Diffusion Maps Algorithms	271

14.2.1	Dmaps DR Algorithm Description	271
14.2.2	Dmaps Algorithm of Graph-Laplacian Type	272
14.2.3	Dmaps Algorithm of Laplace-Beltrami Type	274
14.2.4	Dmaps Algorithm of Self-tuning Type	276
14.3	Implementation of Dmaps for DR	279
14.3.1	Implementation of Dmaps of Graph-Laplacian Type	280
14.3.2	Implementation of Dmaps of Laplace-Beltrami Type	284
14.3.3	Implementation of Dmaps of Self-turning Type	286
14.3.4	Conclusion	287
14.4	Diffusion Maps and Multiscale Diffusion Geometry	287
14.4.1	Construction of General Diffusion Kernels	287
14.4.2	Diffusion Distances	289
14.4.3	Diffusion Maps as Feature Extractors	292
14.5	Implementation of Dmaps for Feature Extraction	293
14.5.1	Feature Extracted from 3-dimensional Toroidal Helix	293
14.5.2	Reordering Face Images	293
14.5.3	Image Parameters Revealing	295
14.5.4	Feature Images of Hyperspectral Image Cube	296
References	· · · · ·	298
Chapter 15	Fast Algorithms for DR Approximation · · · · ·	299
15.1	Low-rank Approximation and Rank-revealing	
	Factorization	299
15.1.1	Rank-revealing Factorization	300
15.1.2	Fast Rank-revealing Algorithms	302
15.1.3	Nyström Approximation	307
15.1.4	Greedy Low-rank Approximation	309
15.2	Randomized Algorithm for Matrix Approximation	311
15.2.1	Randomized Low-rank Approximation	311
15.2.2	Randomized Interpolative Algorithm	312
15.2.3	Randomized SVD Algorithm	314
15.2.4	Randomized Greedy Algorithm	315
15.3	Fast Anisotropic Transformation DR Algorithms	316
15.3.1	Fast Anisotropic Transformation	316
15.3.2	Greedy Anisotropic Transformation	318
15.3.3	Randomized Anisotropic Transformation	319

15.3.4	Matlab Code of FAT Algorithms	320
15.4	Implementation of FAT Algorithms	324
15.4.1	FAT DR of Artificial Surfaces	324
15.4.2	Application of FAT to Sorting Image Datasets	330
15.4.3	Conclusion	331
15.5	Justification	332
15.5.1	Main Proof of Theorem	332
15.5.2	Lemmas Used in the Proof	333
References	336
Appendix A	Differential Forms and Operators on Manifolds	339
A.1	Differential Forms on Manifolds	339
A.2	Integral over Manifold	341
A.3	Laplace-Beltrami Operator on Manifold	345
Index	349

Acronyms

CMDS	classical multidimensional scaling
CSDP	C semidefinite programming
DR	dimensionality reduction
DWT	discrete wavelet transform
EM-PCA	expectation-maximization principle component analysis
FAT	fast anisotropic transformation
FFT	fast Fourier transform
GAT	greedy anisotropic transformation
HEEL	Hessian locally linear embedding
HSI	hyperspectral image
IRAT	interpolative randomizing anisotropic transformation
LLE	locally linear embedding
LMVU	landmark maximum variance unfolding
LTSA	local tangent space alignment
MDS	multidimensional scaling
MVU	maximum variance unfolding
PCA	principle component analysis
PRAT	projective randomizing anisotropic transformation
RAT	randomizing anisotropic transformation
RP	random projection
SDE	semidefinite embedding
SDP	semidefinite programming
SLLE	supervised locally linear embedding
SVD	singular value decomposition
Dmaps	diffusion maps
Isomaps	isometric maps
JL-embedding	Johnson and Lindenstrauss embedding
LB operator	Laplace Beltrami operator
Leigs	Laplacian eigenmaps
i.i.d.	independent and identically distributed
o.n.	orthonormal

xviii Acronyms

o.g.	orthogonal
psd	positive semidefinite
pd	positive definite

Symbols

$\mathfrak{D}_{m,n}$	real $m \times n$ diagonal matrices
\mathfrak{D}_n	real $n \times n$ diagonal matrices
$\mathfrak{M}_{m,n}$	real $m \times n$ matrices
$\mathfrak{M}_{m,n}(r)$	real $m \times n$ matrices of rank r
\mathfrak{M}_n	real $n \times n$ matrices
$\mathfrak{M}_n(r)$	real $n \times n$ matrices of rank r
$\mathfrak{O}_{m,n}$	real $m \times n$ matrices with o.n. columns if $m \geq n$, and with o.n. rows if $m < n$
\mathfrak{O}_n	real $n \times n$ o.g. matrices
\mathfrak{S}_n	$n \times n$ psd matrices
$S\mathfrak{P}_n(r)$	$n \times n$ psd matrices of rank r
\mathfrak{P}_n	$n \times n$ pd matrices
$\mathfrak{R}_{m,n}$	$m \times n$ random matrices
$\mathfrak{R}_{m,n}^1$	$m \times n$ Gaussian random matrices, random matrices of Type 1
$\mathfrak{R}_{m,n}^2$	$m \times n$ random matrices of Type 2
$\mathfrak{R}_{m,n}^3$	$m \times n$ random matrices of Type 3
\mathbb{R}^n	space of real n -vectors, n -dimensional Euclidean space
\mathbb{R}_+	nonnegative real numbers
\mathbb{Z}	integer numbers
\mathbb{Z}_+	nonnegative integer numbers
\mathbb{N}	natural numbers

Chapter 1 Introduction

Abstract This chapter gives an overview of the book. Section 1 briefly introduces high-dimensional data and the necessity of dimensionality reduction. Section 2 discusses the acquisition of high-dimensional data. When dimensions of the data are very high, we shall meet the so-called curse of dimensionality, which is discussed in Section 3. The concepts of extrinsic and intrinsic dimensions of data are discussed in Section 4. It is pointed out that most high-dimensional data have low intrinsic dimensions. Hence, the material in Section 4 shows the possibility of dimensionality reduction. Finally, Section 5 gives an outline of the book.

1.1 Overview of Dimensionality Reduction

In our world, many objects can only be electronically represented with high-dimensional data—speech signals, images, videos, text documents, hand-writing letters and numbers, fingerprints, and hyperspectral images, etc. We often need to analyze a large amount of data and process them. For instance, we often need to identify a person’s fingerprint, to search text documents by keywords on the Internet, to find certain hidden patterns in images, to trace objects from videos, and so on. To complete these tasks, we develop systems to process data. However, due to the high dimension of data, a system directly processing them may be very complicated and unstable so that it is infeasible. In fact, many systems are only effective for relatively low dimensional data. When the dimensions of data are higher than the tolerance of such a system, they cannot be processed. Therefore, in order to process high-dimensional data in the systems, dimensionality reduction becomes necessary. The following paragraphs present the typical cases where dimensionality reduction plays an important role.

Fingerprint identification

A fingerprint is an impression of the friction ridges of all or any part of the finger. A digitalized fingerprint is a digital image of the friction ridges. Assume

the resolution of the image is 64×64 so that the finger print has the dimension 4,096. A dataset of fingerprints may consist of more than one million samples. Fingerprint identification (sometimes referred to as dactyloscopy [1]) is the process of comparing questioned and known friction skin ridge impressions from fingers to determine if the impressions are from the same finger. It is customary to use $m \times n$ to denote a resolution of m pixels wide and n pixels high, where m and n are positive integers.

Face recognition

In face recognition, a database consists of a great number of person's faces of the same resolution, say 256×256 . Without dimensionality reduction, a face recognition/classification system would process 65,536 dimensional real vectors, which are usually transformed from the face images by row concatenation. The dimension is too high to be processed in regular recognition systems. Therefore, we need to reduce the data dimension.

Hyperspectral image analysis/processing

Hyperspectral sensors are used to collect geological/geographical imagery data as a set of images of the same scene, with each image representing a range (also called spectral band) of 5-10 nm (nanometers) of the electromagnetic spectrum. In general, a set of such hyperspectral images contains hundreds of narrow (adjacent) spectral bands of electromagnetic radiation between 350 nm and 3500 nm. These images are combined to form a three-dimensional hyperspectral imagery cube (HSI cube), where images reside on the (2-d) spatial domain while the spectral radiance curve is represented in the spectral domain. Depending on different tasks, either a band image or a spectral radiance curve can be considered as a vector for hyperspectral image analysis/processing.

Text document classification/search

Assume that a pre-set dictionary of keywords consists of n words. Each text document is assigned an n -dimensional vector whose components are the frequency of the keywords occurring. The classifiers and search systems process such vectors. When n is large, dimensionality reduction is needed for a fast and stable search or classification.

Data visualization

Data with dimensions greater than 3 does not have visual presentation. To visualize the features of such data, we reduce their dimensions to 2 or 3.

Data feature extraction

High dimensional data are usually heavily redundant. Important information or features are hidden. We need to reduce the dimension of data to extract the required information or features.

These examples show that in many occasions, it is useful or even necessary to reduce the high-dimensional data to a lower one, keeping as much of the

original information as possible, so that the new low-dimensional data can be effectively used either in processing systems, or for better understanding. Figure 1.1 summarizes the role of dimension reduction.

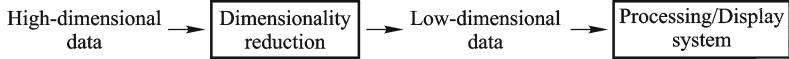


Fig. 1.1 The role of dimensionality reduction. High-dimensional data is first reduced to low-dimensional data, which will be further used either in a processing system for data analysis/processing or for displaying the required information.

1.2 High Dimension Data Acquisition

In dimensionality reduction (DR), an object is usually converted into a vector (also called a point) and then the object set becomes a dataset consisting of vectors with the same dimension. Geometrically, a dataset is shown as a point cloud in a Euclidean space. An object set may be converted into various data sets, depending on the goals of the data processing. We give several examples to illustrate how to make such conversions.

1.2.1 Collection of Images in Face Recognition

Face recognition algorithms are based on facial databases. A typical facial database contains a great number of facial images of many people. The images of each person may be taken under different illumination conditions, different poses, with different facial expressions, and at different time. In a database, all facial images have the same size (which is usually called resolution). Figure 1.2 illustrates some facial images extracted from a facial database, which can be downloaded from the web site www.cs.nyu.edu/~roweis/data.html. Some face dada sets often used by researchers can be obtained from www.face-rec.org.

In general, facial images can be either gray or colored. In DR, we convert each facial image to a vector. For instance, each gray image with the resolution $N = m \times n$ can be transformed to an N -dimensional vector by row or column concatenation, and each color image can be converted to a vector of dimension $N = m \times n \times 3$ if it is equally sampled in the three color channels of R, G, and B. Thus, an image data set with k images is represented by a vector set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_k\} \subset \mathbb{R}^N$. If we want to sort the facial images in the set, we need to reduce the dimension N to 2, such that the two coordinates in the DR space represent two parameters, the pose and face expression respectively.



Fig. 1.2 Olivetti faces. The figure displays 100 faces for 10 people randomly chosen from the collections of Olivetti faces. Each person has 10 facial images with different poses and expressions. The resolution for each image is 64.

The data set \mathcal{X} can also be reduced in different ways. For example, assume that the data set \mathcal{X} is chosen as a training set for face recognition, where k is a great number. Assume also that the eigenface method is used for the face recognition. The method creates a few virtual facial images (called *eigenfaces*) that represent each face in the set as their linear combination. Assume that the number of eigenfaces is s ($s \ll k$). In this case, we want to reduce the number of faces from k to s . From the statistical point of view, we may consider the data set \mathcal{X} as a k dimensional random vector, in which each random variable takes its sample from an image. Then the eigenface method reduces the dimension of the random variable from k to s . We usually required that the new s -dimensional random vector has independent components.

1.2.2 Handwriting Letters and Digits

Converting handwriting letters and digits into vectors is quite similar to the conversion of facial images. Assume that the resolution of an image of a handwriting letter is $N = m \times n$. Then it is converted to an N dimensional binary vector by row or column concatenation. Usually in an original image of a handwriting letter or digit, background is white having the binary value 1, and a letter or digit is black having the value 0. In order to get a sparse representation, when it is converted into a binary vector in processing, the values 0 and 1 are interchanged. Therefore, a set of images of handwriting letters (or digits) is converted to a vector set in the N -dimensional space \mathbb{R}^N . Figure 1.3 illustrates a set of handwriting digits, where the background is black (having the value 0), shown as a negative. In processing, the set is often represented as an $N \times k$ matrix, where each column represents an image of a handwriting letter (or a digit), while k is the number of the images



Fig. 1.3 Handwriting digits. The figure displays 100 handwriting digits. Each image is shown as a negative for its sparse representation.

in the set. Similar to the discussion about the facial images, there are two main tasks on the image set of handwriting letters and digits: recognition and sorting. To sort such a set by two parameters, we shall reduce the dimension of the image vectors from N to 2, while in recognition, we may reduce the number of the images from k to $s \ll k$.

1.2.3 Text Documents

Keyword searching is a common search method used on the Internet. To sort documents from the search result, we often convert a text document into a term-frequency vector in the following way. First, we create a keyword dictionary, which contains n keywords. Then we count the occurrences of each keyword in a document, creating an n -dimensional term-frequency vector for the document. During the vector conversion for a set of documents, we deploy a few filtering methods to omit empty documents, remove common terms, and sometimes stem the vocabulary. For example, the toolkit developed by McCallum [2] can remove the common terms, and the algorithm developed by Porter [3] can be used to stem vocabulary. Each document vector is then normalized to a unit vector before further processing. The text document data of four newsgroups: `sci.crypt`, `sci.med`, `sci.space`, and `soc.religion.christian`, and the data Reuters-21578: Text Categorization Test Collection Distribution 1.0 can be obtained from www.daviddlewis.com/resources/testcollections/reuters21578/.

1.2.4 Hyperspectral Images

Hyperspectral images are acquired by hyperspectral sensors, which collect geological/geographical imagery data as a set of images of the same scene, with each image representing a range (also called spectral band) of 5-10 nm (nanometers) of the electromagnetic spectrum. In general, a set of such hyperspectral images contains hundreds of narrow (adjacent) spectral bands of electromagnetic radiation between 350 nm and 3500 nm. Hyperspectral images are often combined to form a three-dimensional hyperspectral imagery cube (or HSI cube) for image processing and analysis (see Fig. 1.4).

Let $f : C = [a, b] \times [c, d] \times [s_1, s_2] \subset \mathbb{R}^3 \rightarrow \mathbb{R}$ denote an HSI cube, where the spatial domain $[a, b] \times [c, d]$ is called the *sensing area* and the interval $[s_1, s_2]$ is called the *range of the spectral band*. At each fixed position $(x_0, y_0) \in [a, b] \times [c, d]$, the function $f_0(s) = f(x_0, y_0, s)$ is called the *raster cell*, and the graph of a raster cell is called *spectral radiance curve* (or simply called *spectral curve*). Material components of the object at a raster cell can be identified from the spectral curves. Hyperspectral sensors are used

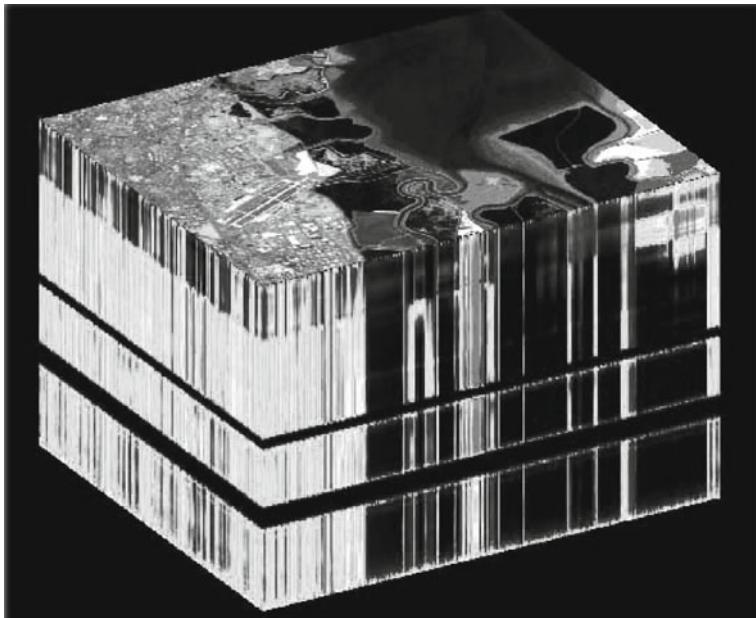


Fig. 1.4 Hyperspectral images. Hyperspectral images are often represented as a three-dimensional cube called a hyperspectral imagery cube or HSI cube. In the cube, each vertical line represents a spectral vector while each horizontal section is a band image.

to scan an area of the spatial domain $[a, b] \times [c, d]$ to produce HSI images with more than one sample—a sample per month, for example. Recently, high-resolution sensors can capture a square-meter area for each raster.

The precision of hyperspectral sensors is typically measured in terms of both spectral resolution (which is the width of each band of the captured spectrum) and spatial resolution (which is the size of a raster in an HSI). Since a hyperspectral sensor can collect a large number of fairly narrow bands, it is possible to identify objects even if they are only captured in a handful of pixels. It is important to point out that spatial resolution contributes to the effectiveness of spectral resolution. For example, if the spatial resolution is too low, then multiple objects might be captured within the same raster, making it difficult to identify the objects of interest. On the other hand, if a pixel covers a too small area, then the energy captured by the sensor-cell could be so low that the signal-to-noise ratio is decreased too much to preserve the reliable features. To achieve high spatial resolution imagery (HRI), a high resolution black-and-white (or panchromatic) camera is integrated in the HSI system. The typical spatial resolution of HSI systems without HRI cameras is one square-meter per pixel, and it can reach a few square-inches per pixel if the systems are integrated with high-quality HRI cameras.

Some HSI data are free to use in research, while others are for commercial

use only. The following are some web sites in the USA providing free HSI data for research:

- Jet Propulsion Laboratory, California Institute of Technology: aviris.jpl.nasa.gov/html/data.html
- L. Biehl: <https://engineering.purdue.edu/~biehl/MultiSpec/hyperspectral.html> (Purdue University)
- Army Geospatial Center: www.agc.army.mil/Hypercube

In DR, we represent an HSI cube as a matrix, where each column is a spectral vector, and each row is a vector, obtained from a band image by row or column concatenation. For example, if an HSI cube has s bands and n raster cells, then the dimension of the matrix is $s \times n$. The HSI data can be considered as either a set of spectral vectors in \mathbb{R}^s or a set of band images (as a vector in \mathbb{R}^n), depending on the purpose of the processing.

The raw HSI data has special formats. A common HSI data includes a head file (with the extension `.hdr`), a wavelength description file (with the extension `.wvl`) and a data file (without extension, or with the extension `.dat`). All of these files have the same file name. The head file describes how to read the raw data in the data file, and the wavelength description file provides the wavelength information for each band image.

1.3 Curse of the Dimensionality

When we deal with high-dimensional data, we meet the *curse of the dimensionality*. DR is a way to avoid it. The term *curse of the dimensionality* was first coined by Bellman [4], referring to the fact that in the absence of simplifying assumptions, the sample size required to estimate a function of several variables to a given degree of accuracy (i.e., to get a reasonably low variance estimate) grows exponentially with the increasing number of variables. For example, the grid of spacing $1/10$ on the unit cube in \mathbb{R}^D consists of 10^D points, which grows exponentially as the dimension D approaches infinity. Another example comes from the estimate of local average. Most density smoothers are based on local average of the neighboring observations. In order to find enough neighbors in high-dimensional spaces, multivariate smoothers have to reach out farther, losing the locality.

A related fact responsible for the curse of the dimensionality is the empty space phenomenon [5]. That is, high-dimensional spaces are inherently sparse. For a uniform distribution of the unit sphere in \mathbb{R}^{10} , the probability of finding a point at a distance $\sqrt{0.9}$ from the center is only 0.35, and a standard 10-dimensional normal distribution $N(0, \mathbf{I})$ has only 0.02% of the mass in the unit sphere. Therefore, contrarily to our intuition, in high-dimensional distributions the tails are much more important than one-dimensional cases. Another problem caused by the curse of the dimensionality is that if there are linear correlations in the data, the optimal mean integrated squared error

will be very large in the estimate of the data density even if the sample size is arbitrary large [6].

We now illustrate the curse of dimensionality in the following subsections, which are largely inspired by Chapter 1 in [5] (also see Section 2.2 in [7] and Appendix I in [8]).

1.3.1 Volume of Cubes and Spheres

Let $V_{sph}^D(r)$ denote the *volume of the sphere* with radius r in \mathbb{R}^D . We have

$$V_{sph}^D(r) = \frac{\pi^{D/2} r^D}{\Gamma(D/2 + 1)}, \quad (1.1)$$

where $\Gamma(x)$ is the gamma function. Let $V_{cube}^D(r)$ denote the *volume of the cube* of side $2r$ in \mathbb{R}^D . We have

$$V_{cube}^D(r) = (2r)^D. \quad (1.2)$$

Although both volumes have the same power growth order D with respect to the linear size r , their constants are very different: The ratio $V_{sph}^D(r)/V_{cube}^D(r)$ approaches zero as the dimension n increases.

$$\lim_{D \rightarrow \infty} \frac{V_{sph}^D(r)}{V_{cube}^D(r)} = 0.$$

That is, when dimension increases, the volume of the cube concentrates more in its corners and less in the inscribed sphere.

1.3.2 Volume of a Thin Spherical Shell

We now consider the volume of a thin spherical shell between two concentric spheres of respective radii r and R , ($R > r$). By (1.1), the relative volume of the thin shell is

$$\frac{V_{sph}^D(R) - V_{sph}^D(r)}{V_{sph}^D(R)} = 1 - \left(\frac{r}{R}\right)^D.$$

When the dimension D increases, the relative volume of the shell tends to 1, meaning that the shell contains almost all the volume of the larger sphere no matter how thin the shell is. Hence, virtually all the content of a D dimensional sphere concentrates on its surface, which is only a $(D - 1)$ dimensional manifold. Therefore, if the points of a data distribute uniformly over either a sphere or a cube, most of them fall near the boundary of the object.

1.3.3 Tail Probability of the Multivariate Gaussian Distributions

The preceding examples show that if data distribute uniformly over a cube in a high-dimensional Euclidean space, most of the data will not be in the inscribed sphere. In this subsection, we show that in high-dimensional cases, the tail probability of isotropic Gaussian distributions can be quite large. It is known that the probability density function (pdf) of an isotropic Gaussian distribution in \mathbb{R}^D is

$$G_\sigma(\mathbf{x}) = \frac{1}{\sqrt{(2\pi\sigma^2)^D}} \exp\left(-\frac{\|\mathbf{x} - \mathbf{a}\|^2}{2\sigma^2}\right),$$

where $\mathbf{x} \in \mathbb{R}^D$, \mathbf{a} is the D -dimensional mean, and σ is the standard deviation. If the distribution has zero mean and unit variance, then its pdf is reduced to

$$G(\mathbf{x}) = K(r) = \frac{1}{\sqrt{(2\pi)^D}} \exp\left(-\frac{r^2}{2}\right), \quad r = \|\mathbf{x}\|, \quad (1.3)$$

which indicates that the equiprobable contours for the distribution are the spheres centered at the origin. Let B_r denote the sphere $\{\mathbf{x} \in \mathbb{R}^D; \|\mathbf{x}\| \leq r\}$, and S_r denote its boundary, which is a $(D-1)$ dimensional sphere. Then the volume of S_r is

$$V(S_r) = \frac{2\pi^{D/2} r^{D-1}}{\Gamma(D/2)}.$$

By (1.3), the probability that a point is in the sphere B_r is

$$\Pr[\|\mathbf{x}\| \leq r] = \frac{\int_0^r V(S_r) K(r) dr}{\int_0^\infty V(S_r) K(r) dr}.$$

To compare the tail probabilities of different dimensional Gaussian distributions, we compute $\Pr[\|\mathbf{x}\| \geq 2]$ for $D = 1, 2, 5, 10, 20, 100$ respectively and display the results in Table 1.1.

Table 1.1 The tail probabilities of the multivariate Gaussian distributions for different dimensions: the tail probability is computed outside the sphere of radius 2.

n	1	2	5	10	20	100
Probability	0.04550	0.13534	0.54942	0.94734	0.99995	1.00000

The table shows the probability mass of a multivariate Gaussian rapidly migrates into the tails, as the dimension increases. In high dimensions, say larger than 20, the entire sample will be in the tails.

1.3.4 Diagonals of Cube

We now consider the angle between a diagonal and a side of the cube $[-1, 1]^D \subset \mathbb{R}^D$, assuming D is large. Let a diagonal vector from the center to a corner be denoted by \mathbf{v} . Then \mathbf{v} is one of the 2^D vectors of the form $(\pm 1, \dots, \pm 1)^T$. The angle between a diagonal vector \mathbf{v} and a coordinate axis \mathbf{e}_i is given by

$$\cos \theta = \left\langle \frac{\mathbf{v}}{\|\mathbf{v}\|}, \mathbf{e}_i \right\rangle = \frac{\pm 1}{\sqrt{D}},$$

which tends to zero, when D grows. Thus, the diagonals are nearly orthogonal to all coordinate axes for large D . In visualization, high-dimensional data are plotted by using their two coordinates. However, the pairwise scatter plot can be misleading. Indeed, any data cluster lying near a diagonal in the space will be plotted near the origin, while a cluster lying near a coordinate axis will be plotted as intuitively expected.

1.3.5 Concentration of Norms and Distances

Another problem encountered in high-dimensional spaces regards the weak discrimination power of a metric, that is, the distribution of the norms in a given distribution over the points concentrates. The following theorem in [9] explains this concentration phenomenon.

Theorem 1.1. *Let $\mathbf{x} = [x_1, \dots, x_D]' \in \mathbb{R}^D$ be a random vector whose components $x_k, 1 \leq k \leq D$, are independent and identically distributed (i.i.d.), with a finite eighth order moment. Let $\mu = E\{x_k\}$ be the common mean of all components x_k of \mathbf{x} , and $\mu_j = E((x_k - \mu)^j)$ be their common central j^{th} moment. Then the mean $\mu_{\|\mathbf{x}\|}$ and the variance $\sigma_{\|\mathbf{x}\|}^2$ of the Euclidean norm of \mathbf{x} are*

$$\begin{aligned} \mu_{\|\mathbf{x}\|} &= \sqrt{aD - b} + O(D^{-1}), \\ \sigma_{\|\mathbf{x}\|}^2 &= b + O(D^{-1/2}), \end{aligned} \tag{1.4}$$

respectively, where a and b are the parameters defined by

$$\begin{aligned} a &= \mu^2 + \mu_2, \\ b &= \frac{4\mu^2\mu_2 - \mu_2^2 + 4\mu\mu_3 + \mu_4}{4(\mu^2 + \mu_2)}. \end{aligned}$$

Equation (1.4) illustrates that when the dimension D increases to ∞ , the mean of the norms of random vectors has the growth order $1/2$, as expected, but the variance of the norms is near a constant b . This means that random vectors are nearly normalized in high dimensions. More precisely, thanks to

Chebychev's inequality, we have

$$\Pr(|\|\mathbf{x}\| - \mu_{\|\mathbf{x}\|}| \geq \varepsilon) \leq \frac{\sigma_{\|\mathbf{x}\|}^2}{\varepsilon^2}.$$

Hence, the probability of norm $\|\mathbf{x}\|$ falling in the interval $(\mu_{\|\mathbf{x}\|} - \varepsilon, \mu_{\|\mathbf{x}\|} + \varepsilon)$ is only dependent on ε , but independent of the dimension D . Since $\mu_{\|\mathbf{x}\|}$ increases to $D^{1/2}$, when D is very large, replacing $\|\mathbf{x}\|$ by $\mu_{\|\mathbf{x}\|}$ only makes a negligible relative error. Therefore, high-dimensional random i.i.d. vectors are distributed close to the surface of a sphere of radius $\mu_{\|\mathbf{x}\|}$: not only successive drawings of such random vectors yield almost the same norm, but also the Euclidean distance between any two vectors is approximately constant.

In practice, the concentration phenomenon makes the nearest-neighbor search problem difficult to solve in high dimensions [10, 11, 12, 13].

1.4 Intrinsic and Extrinsic Dimensions

In practice, high-dimensional data are often not *truly high-dimensional*. It is a consensus in the high-dimensional data analysis community that the points of high-dimensional data usually reside on a much low-dimensional manifold. Assume that a data set $\mathcal{X} = \{\mathbf{x}_\alpha\}_{\alpha \in A}$ resides on an s -dimensional manifold M that is embedded in \mathbb{R}^D : $M \subset \mathbb{R}^D$. Then we call D the *extrinsic dimension* of \mathcal{X} and s the *intrinsic dimension* of \mathcal{X} . From the viewpoint of statistics, \mathcal{X} can be considered as a sample set of a random vector \mathbf{X} in \mathbb{R}^D . If \mathbf{X} is governed by s -independent variables, that is, there are a random vector $\mathbf{Y} \in \mathbb{R}^s$ and an invertible analytic function $f: \mathbb{R}^s \rightarrow \mathbb{R}^D$ such that $f(\mathbf{Y}) = \mathbf{X}$, then the random vector \mathbf{X} is said to have intrinsic dimension s . The low intrinsic dimensionality of high-dimensional data is the key to the feasibility of dimensionality reduction. Due to the low intrinsic dimension of data, we can reduce the (extrinsic) dimension without losing much information for many types of real-life high-dimensional data, avoiding many of the curses of dimensionality. In one sense, dimensionality reduction is processing to find a certain parameterization of the manifold which the points of data reside on.

1.4.1 Intrinsic Dimension Estimation

As we discussed above, intrinsic dimension estimation of data is very important for high-dimensional data analysis. However, it is a quite difficult problem, particularly for finite sets. Theoretically, a finite data set can be embedded in many manifolds of various dimensions, depending on the geometric structures defined on the data. The data set in Fig. 1.5 is a data set on plane. However it can be embedded in either the plane or the one-dimensional

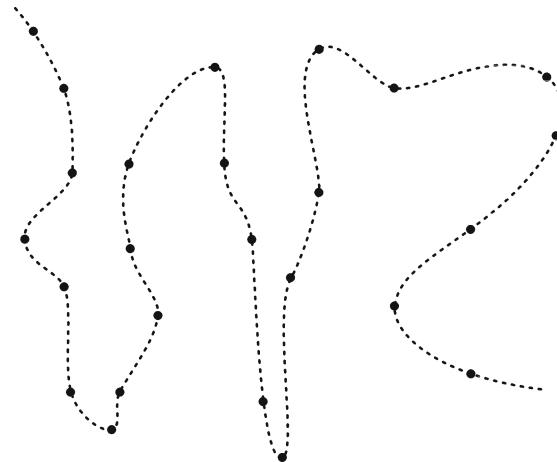


Fig. 1.5 A data set on the plane can be also embedded in a one-dimensional curve.

curve shown in the figure. The simplest manifold is linear, usually called hyperplane. For a data set $\mathcal{X} \subset \mathbb{R}^D$, among all hyperplanes $S \subset \mathbb{R}^D$ such that $\mathcal{X} \subset S$, there is a unique one that has the lowest dimension, say, s . Then s is the linear intrinsic dimension of the data \mathcal{X} . The orthogonal (shortly, o.g.) embedding from \mathbb{R}^s to S can be represented as the linear operator $T : \mathbb{R}^s \rightarrow S :$

$$\mathbf{x} = T(\mathbf{y}) = \mathbf{U}\mathbf{y} + \mathbf{x}_0,$$

where $\mathbf{U} \in \mathfrak{O}_{D,s}$ is an o.g. transformation and \mathbf{x}_0 is the nearest point on S to the origin. If S is a subspace of \mathbb{R}^D , then $\mathbf{x}_0 = 0$. The inverse of the o.g. embedding T provides an s -dimensional parametric representation of the data set \mathcal{X} :

$$\mathcal{Y} = \{\mathbf{y} \in \mathbb{R}^s : \quad \mathbf{y} = T^{-1}(\mathbf{x}), \quad \mathbf{x} \in \mathcal{X}\}.$$

Without loss of generality, we may assume $\mathbf{x}_0 = \mathbf{0}$. Write $\mathbf{U} = [\mathbf{u}_1 \cdots \mathbf{u}_s]$. The set $\mathcal{U} = \{\mathbf{u}_1, \dots, \mathbf{u}_s\}$ is an o.n. (orthonormal) basis of the subspace S . Therefore, \mathbf{y} is the coordinates of $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ under the o.n. basis \mathcal{U} . It is clear that the Euclidean distances between the points of the data set \mathcal{Y} preserve those of the data set \mathcal{X} . Hence, when a data set is equipped with the (global) Euclidean geometry, its intrinsic dimension is a linear one, and the mapping from \mathcal{X} to \mathcal{Y} provides a linear DR. Recovering the original data \mathcal{X} from its DR data \mathcal{Y} is very simple. Linear DR is very useful for those applications, in which the data recovering is crucial. In practice, noise and irrelevant components exist in data. To eliminate their effect, Principal Component Analysis (PCA) is a powerful tool in discovering the linear dimensionality of data. We shall discuss PCA in more detail in Chapter 5. However, when a data set has a nonlinear geometric structure, the linear

intrinsic dimension cannot reflect its true dimension since a nonlinear geometry endows the data with a (nonlinear) manifold structure, not a linear one. By the manifold theory, if a manifold $M \subset \mathbb{R}^D$ has dimension $s < D$, then the neighborhood of each point of M is isomorphic to \mathbb{R}^s . Therefore, we may estimate the intrinsic dimension of a data from its neighborhood structure. The manifold dimension can also be expressed as the classical concept of *topological dimension*. To give its definition, we need some basic notions in topology. Given a topological space S , the open covering of a set $\mathcal{X} \subset S$ is a collection \mathcal{O} of open subsets in S whose union contains \mathcal{X} . A refinement of a covering of \mathcal{X} is another open covering \mathcal{O}' such that each set in \mathcal{O}' is a subset of some set in \mathcal{O} . It is known that an s -dimensional set \mathcal{X} can be covered by open spheres such that each point belongs to at most $s + 1$ open spheres. Thus, we define the topological dimension of a set as follows.

Definition 1.1. A subset \mathcal{X} of a topological space S is said to have topological dimension s_{topo} (also called Lebesgue covering dimension) if every covering \mathcal{O} of \mathcal{X} has a refinement \mathcal{O}' such that every point of \mathcal{X} is covered by at most $s + 1$ open sets in \mathcal{O}' , and s is the smallest among such integers.

There is a technical difficulty to estimate the topological dimension for a finite sample. Hence, practical methods use various other definitions of the intrinsic dimension. It is common to categorize intrinsic dimension estimating methods into two classes, projection techniques and geometric approaches.

The manifold projection methods are widely used in DR. In these methods, a neighborhood structure is first defined on the data. If each neighborhood can be projected to an s -dimensional space within a tolerance, then the intrinsic dimension of the data is s , and we can reduce the dimension to s without losing much information. The manifold projection methods can produce the dimension-reduced data useful for improving performance in classification and other applications. However, in these methods, the geometric structure of the data has to be given in advance via the definition of the neighborhood. If the size of the neighborhood is too small, then the geometry on the data will cause important data features to be collapsed. On the other hand, if the neighborhood is too large, which creates a large intrinsic dimension, then the manifold projections could become noisy and unstable. In general, there is no consensus on how this dimension should be determined. Fortunately, in a task of DR, the reduced dimension (i.e., the parameters for the random vector) is preset according to its purpose. Therefore, manifold projections favor the construction of DR data much better than the estimation of the intrinsic dimension.

The geometric approaches that exploit the intrinsic dimension of a data set are often based on fractal dimensions or the nearest neighbor (NN) distances. Although the geometric methods also utilize the geometry of data neighborhood, they do not require the parameters as input. Most of these methods estimate the intrinsic dimension by counting volume growth rate of an expanding sphere on the manifold. Several popular methods are briefly

introduced in the following.

1.4.2 Correlation Dimension

The correlation dimension [14, 15] perhaps is the most popular fractal dimension. It estimates the dimension of data based on the simple fact: The volume of an s -dimensional sphere of the radius r is proportional to r^s . Hence, if M is an s -dimensional manifold in \mathbb{R}^D , then the volume of the r -neighborhood of a point $\mathbf{x} \in M$ is $\sim r^s$. We may apply this result to the estimation of the dimension of a discrete set: Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ be a data set whose points are uniformly distributed on an s -dimensional manifold in \mathbb{R}^D . Then the number of pairs of points closer to each other than to r is proportional to r^s . Therefore, the correlation dimension of a countable data set $\mathcal{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_n, \dots\} \subset \mathbb{R}^D$ can be defined as follows.

Definition 1.2. Let $M \subset \mathbb{R}^D$ be an s -dimensional manifold. Assume that a countable set $\mathcal{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_n, \dots\}$ resides on M . Write $\mathcal{Z}_n = \{\mathbf{z}_1, \dots, \mathbf{z}_n\} \subset \mathcal{Z}$. Define

$$C_n(r) = \frac{2}{n(n-1)} \sum_{i=1}^n \sum_{j=i+1}^n I_{\{d(z_i, z_j) < r\}}, \quad z_i, z_j \in \mathcal{Z}_n,$$

where I_A is the indicator function of an event A , such that

$$I_A = \begin{cases} 1, & \text{if } A \text{ is true} \\ 0, & \text{if } A \text{ is false} \end{cases},$$

and

$$C(r) = \lim_{n \rightarrow \infty} C_n(r).$$

The correlation dimension of \mathcal{Z} is defined by

$$d_{corr} = \lim_{r \rightarrow 0} \frac{\log C(r)}{\log(r)},$$

provided the limit exists.

In practice, since the data is finite, the limit cannot be achieved. The estimation procedure usually consists of plotting $\log C_n(r)$ versus $\log r$ and measuring the slope $\frac{\partial \log C_n(r)}{\partial \log r}$ of the linear part. A line fitting technique, for example, the least squares, is often used to find the slope [16, 17]. Finally, in the definition of the correlation dimension, the distance is not necessary to be Euclidean, it can be any distance of a metric space.

1.4.3 Capacity Dimension

If the data distribution on a manifold is not uniform, the correlation dimension can severely underestimate the topological dimension. To resolve this problem, we turn to the capacity dimension, which is another member of the fractal dimension family. For the formal definition, we need to introduce some concepts. Given a metric space \mathcal{X} with distance metric $d(\cdot, \cdot)$, the r -covering number $N(r)$ of a subset $\hat{X} \subset \mathcal{X}$ is defined as the minimum number of open spheres $B(\mathbf{x}_0; r) = \{\mathbf{x} \in \mathcal{X} : d(\mathbf{x}_0, \mathbf{x}) < r\}$ whose union is a covering of \hat{X} . The following definition is based on the observation that the covering number $N(r)$ of an s -dimensional set is proportional to r^s .

Definition 1.3. The capacity dimension of a subset \hat{X} of a metric space \mathcal{X} is

$$d_{capa} = \lim_{r \rightarrow 0} \frac{\log N(r)}{\log r}.$$

The principal advantage of the capacity dimension over the correlation dimension is that the capacity dimension does not depend on the data distribution on the manifold. Moreover, if both d_{capa} and d_{topo} exist, then two dimensions agree with each other. An efficient capacity dimension estimating method is given in [18].

1.4.4 Multiscale Estimation

When the number of sample points is not exponential in the intrinsic dimension and noise is added to the manifold, the above methods sometimes may not estimate the intrinsic dimension well. The authors of [19] introduce the multiscale estimation method, which only requires a number of points essentially proportional to the intrinsic dimensionality. As above, let M be a smooth s -dimensional Riemannian manifold with bounded curvature, isometrically embedded in \mathbb{R}^D , and $\mathcal{X} \subset M$ is the data set that is uniformly distributed on M . The observations $\tilde{\mathcal{X}}$ of \mathcal{X} are noisy samples so that $\tilde{\mathcal{X}} = \{\mathbf{x}_i + \sigma \boldsymbol{\eta}_i\}_{i=1}^n$, where $\boldsymbol{\eta}_i$ are i.i.d. samples from $\boldsymbol{\eta} \sim N(0, 1)$ and $\sigma > 0$. Then we represent a set of n points in R^D by a $D \times n$ matrix, whose (i, j) entry is the i th coordinate of the j th point. Particularly, \mathbf{X} (and $\tilde{\mathbf{X}}$) will be used to denote both the data and the associated $D \times n$ matrices. Let \mathbf{N} denote the noise matrix of the $\boldsymbol{\eta}_i$'s. Then $\tilde{\mathbf{X}} = \mathbf{X} + \mathbf{N}$. The multiscale algorithm estimates the pointwise dimensionality of the data set \mathbf{X} as follows. For a point $\mathbf{z} \in M$, let $\tilde{X}(r) = \tilde{\mathbf{X}} \cap B_{\mathbf{z}}(r)$, where $B_{\mathbf{z}}(r)$ is the sphere of radius r centered at \mathbf{z} . Assume that $\{\sigma_i^r\}_{i=1}^m$ are the singular values of the matrix $\tilde{\mathbf{X}}(r)$. Let them be classified according to the growth rate in r , and then according to the growth rates, the following three sets have particular meanings: The set containing all singular values with linear growth rate in r

is corresponding to the tangent plane at \mathbf{z} , the set of all singular values with quadratic growth rate is corresponding to curvature, and the set of all singular values that do not grow is corresponding to noise. Then the dimension of the data \mathcal{X} can be estimated by the cardinal number of the first set above. The examples of this method can be found in [19, 20].

There are other estimating methods in literature. For example, a method based on the convergence rates of a certain U-statistic on the manifold is introduced in [21], an estimator based on geodesic entropy graphs can be found in [22], and the maximum likelihood estimation is in [23]. The detailed discussion about the estimation of intrinsic dimension is beyond the scope of this book. The readers who are interested in this topic may refer to the references above.

1.5 Outline of the Book

1.5.1 Categories of DR Problems

In general, dimensionality reduction means the following problem: Given an observed high-dimensional dataset $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$, find a low-dimensional dataset $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\} \subset \mathbb{R}^d$ ($d \ll D$) such that certain tasks of data analysis of the high-dimensional data \mathcal{X} can be realized on the low dimensional data \mathcal{Y} within a tolerable error. The DR problems are usually classified into two categories.

Hard dimensionality reduction problems

In hard dimensionality reduction problems, the extrinsic dimensions of the data usually range from hundreds to hundreds of thousands of components, and usually a drastic reduction (possibly of orders of magnitude) is sought out. The components are often repeated measures of a certain magnitude in different points of space or in different instants of time. In this category we would find pattern recognition and classification problems involving images (e.g. face recognition, character recognition, etc.) or speech (e.g. auditory models).

Soft dimensionality reduction problems

In soft dimensionality reduction problems, the extrinsic dimensions of the data usually are not too high—less than a few tens, for example, and the reduction is not very drastic. Typically, the components are observed or measured values of different variables, which have a straightforward interpretation. Most cases of statistical analysis in fields like social sciences, psychology, etc., fall in this category. Multivariate analysis is a main tool for soft DR.

Usually, the main focus of soft DR is not on data reduction, but on data analysis.

The dividing line between these two categories is not very strict. Some techniques, such as PCA, may be used in both of them.

Visualization problems often need the aid of DR, where the high-dimensional data need to be reduced to 2- or 3-dimensions in order to plot it. Several presentation techniques allow one to visualize up to about 5-dimensional data, using colors, rotation, stereography, glyph or other devices, but they lack the appeal of a simple plot. Both soft DR and hard DR methods may reduce high-dimensional data to visible data. To choose a better DR method for the visualization of a particular data is quite dependent on the purpose of the visualization.

1.5.2 Scope of This Book

This book will focus on the *geometric approach* to hard DR problems. This approach adopts manifold embedding technique that embeds a high-dimensional data set into a low-dimensional manifold, by applying eigenvector-optimization on a DR kernel. Hence, it is also called *spectral method*. A manifold embedding technique usually consists of the following steps: For a given observed high-dimensional data, a *neighborhood system* is first designed on it to describe the similarity between the points (i.e., the digitized objects) of the data set. The neighborhood system is formulated by the *weighted graph* mathematically. This formulation equips the data with a *manifold structure*, presenting the data geometry. Each manifold has a coordinate system, which provides a low-dimensional representation of the original data. The *spectral decomposition* of the weight matrix of the graph yields the coordinate representation, called the *DR data*. In this book, the weight matrix is called *DR kernel*.

The first part of this book is devoted to the geometry of data. A brief review of manifold comes first, followed by a preliminary discussion of the spectral graph theory. These are the main tools to reveal the geometry of data. At the end of the first part, we briefly discuss the data models and the formats of the input data (the observed data) and output data (the DR data).

In Part II, linear DR methods are introduced. Besides introducing the deterministic linear DR methods—principal component analysis (PCA) and classical multidimensional scaling (CMDS), we also present the random projection method.

In this book, we are particularly interested in nonlinear DR methods, which constitute Part III, the major part of this book. We now give a brief introduction to these methods.

Isomap

Isometric maps method (Isomaps) [24] seeks to preserve the geodesic distances between points while mapping the observed data into fewer dimensions. Geodesic distance is defined as the length of the shortest path between two points that stays on the manifold surface. Since the data in processing are discrete, the algorithm estimates geodesic distances by finding the shortest graphic path between every pair of points, which can be computed efficiently using the Dijkstra's algorithm. The method preserves the data geometry very well if the data lies on a convex manifold. Isomap produces a dense DR kernel. Its weakness is primarily due to inaccuracies in its estimate of geodesic distance. It tends to produce poor results near the under-sampled regions of the manifold.

Maximum Variance Unfolding (MVU)

Maximum variance unfolding was formerly known as semidefinite embedding (SDE) or semidefinite programming (SDP) [25]. The intuition for this algorithm is based on that when a manifold is properly unfolded, the variance over the points is maximized. This algorithm begins by finding the k -nearest neighbors of each point, then maximizes the variance over the whole data set, preserving the distances between all pairs of neighboring points. MVU preserves the local data geometry very well. It can also be applied to similarity configuration, serving as a nonlinear MDS. However, MVU kernel is dense too. Hence it has a high computational cost. When the dimension of input data is too high, landmark technique is often adopted to reduce the computational time. Some other methods adopt the similar idea as MVU. For example, *manifold sculpting* (MS) [26] uses graduated optimization to find an embedding. It also computes k -nearest neighbors and tries to seek an embedding that preserves relationships in local neighborhoods. It slowly scales variance out of higher dimensions, while simultaneously adjusting points in lower dimensions to preserve those relationships. If the rate of scaling is small, it can find very precise embeddings. The similar idea is also adopted by *locally multidimensional scaling* (LMDS) [27], which performs multidimensional scaling in local regions, and then uses convex optimization to fit all the pieces together.

Locally Linear Embedding (LLE)

Locally linear embedding (LLE) [28], as its name mentions, linearly embeds each neighborhood on the data graph to a low-dimensional space so that the manifold the data resides on is mapped to the same space. LLE begins by constructing k -nearest neighbors of each point, and then computes the barycentric coordinates of the point with respect to its neighbors to describe the local similarity. Finally, the DR data is obtained by preserving the similarity weights. LLE produces sparse DR kernel. Taking the advantage of sparsity, it implements faster than Isomaps and MUV. LLE tends to handle

non-uniform sample densities poorly because there is no fixed unit to prevent the weights from drifting as various regions differ in sample densities.

Local Tangent Space Alignment (LTSA)

LTSA [29] is based on the intuition that when a manifold is correctly unfolded, all of the tangent hyperplanes to the manifold will become aligned. The method is very similar to LLE. It begins by computing k -nearest neighbors of every point, then computes an o.g. basis for the tangent space at every point. The local coordinates associated with the basis produce the weights between the point and its k -nearest neighbors. The weights then derive a sparse DR kernel. LTSA finds an embedding that aligns the tangent spaces by using the eigenvector optimization of the DR kernel.

Laplacian Eigenmaps (Leigs)

Laplacian eigenmaps (Leigs) [30] use spectral technique in DR processing. In Leigs, the high-dimensional data is still assumed to be laid on a low-dimensional manifold. The method first builds a graph that defines a neighborhood system on the data set, then constructs a kernel that approximates the Laplace-Beltrami operator on the manifold. The near-zero eigenfunctions of the Laplace-Beltrami operator are used as the embedding.

Hessian Locally Linear Embedding (HLLE)

Like LLE and Leigs, Hessian LLE (HLLE) [31] also produces a sparse kernel. It tends to construct the approximative Hessian operator on the manifold on which the given data resides so that the DR data is obtained from the near-zero eigenvectors of the operator. It often yields results of higher quality compared with LLE and Leigs, when the given data is smooth. As a trade-off, it has a more costly computational complexity. Besides, it is relatively sensitive to data noise.

Diffusion Maps (Dmaps)

The method of diffusion maps [32] uses the idea that the Neumann heat diffusion operator on a manifold has the same set of eigenvectors as the Laplace-Beltrami operator, but the small eigenvalues of the Laplace-Beltrami operator becomes the largest ones. The method constructs DR kernel that approximates the heat diffusion operator. The DR data then is provided by several leading eigenvectors of the kernel (i.e., the eigenvectors achieving several largest eigenvalues).

In many applications, the dimensions of DR kernels can be very large. That will cause both memory and computational cost problems, and the computational stability problems. To overcome the difficulty caused by the large sizes of DR kernels, we introduce some fast algorithms for spectral decompositions of DR kernels in the last chapter of Part III.

1.5.3 Other Topics Related to This Book

There are other approaches to nonlinear DR such as neural network methods in machine learning. These approaches are closely related to the geometric one. In fact, the methods of manifold embedding is based on unsupervised (manifold) learning. They have very close relation with the neural network methods. Some neural network methods are briefly described in the following.

Self-organizing map (SOM)

Self-organizing map also called self-organizing feature map (SOFM) or Kohonen map, is a type of artificial neural network that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional) discretization of the input space of the training samples. Self-organizing maps are different from other artificial neural networks in that they use a neighborhood function to preserve the topological properties of the input space [33].

Generative topographic map (GTM)

Generative topographic map is a potential alternative to SOMs. GTM explicitly requires a smooth and continuous mapping from the input space to the map space, preserving topology. However, in a practical sense, the measure of topological preservation is lacking [34, 35].

Autoencoders

A completely different approach to nonlinear dimensionality reduction is the use of *autoencoders*, a special kind of feed-forward neural networks. Although the idea of autoencoders is not new, training of the encoders has only recently become possible through the use of restricted Boltzmann machines. Related to autoencoders is the *NeuroScale* algorithm, which uses stress functions inspired by multidimensional scaling and Sammon mappings to learn a nonlinear mapping from the high-dimensional space to the embedded space. The mappings in NeuroScale are based on radial basis function networks.

Data compression

DC has a close relation with DR. It is another technique often used in the high-dimensional data acquisition and processing. Data compression or source coding is the process of encoding data down to size smaller than their normal presentation. Because of the huge size of high-dimensional data, it is necessary to transmit the compressed data instead. There are two compression schemes: lossless and lossy. Lossy compression scheme can reach a higher compression rate than the lossless one, but, as a trade-off, will lose some information. Hence, the quality control for a lossy compression scheme is crucial. There is a close relation between DR and compression: a high-dimensional data can be first reduced to a low-dimensional data, and then compressed further. Since the compressed data still need to be restored to the original one (within

a tolerance), usually only linear DR methods can be used as a pre-process for data compression. The main difference between DR and DC is that the main purpose of DR is efficiency of data processing while DC seeks efficiency behavior of data transmission. The compressed data cannot be used in data processing unless it is restored, whereas DR data can. Recently, compressive sensing (CS) has motivated intensive research activities. This new methodology integrates the compression schemes into the data acquisition so that no additional compression is necessary for the acquired data. The readers interested in compressive sensing may refer to [36–40].

Since these topics are outside the main focus of our subject, they will not be covered in this book. The readers may find them in the above references.

1.5.4 Artificial Surfaces for Testing DR Algorithms

There are many different DR algorithms. Each has its own advantages and drawbacks. In practice, in order to test the validity of a DR method for an application, say, classification, we divide a chosen database into two sets: one is training data, the other is testing data. The training data is used for the development of algorithm, and the testing data for testing the validity of the algorithm. Since the data classes of the testing data are known, the percentage of misclassified objects can be used to measure the effectiveness of the method. Note that the validity of a DR method depends on the data geometry. It is very hard to give a quality comparison between the various methods by the testing above.

Artificially created data sets are also often used to illustrate the advantages and drawbacks of the various methods. The following manifolds are very often used as standard examples in the DR research group. The first manifold is called the Swiss roll, originating from the name of a Swiss-made cake: it is composed of a layer of airy pastry, which is spread with jam and then rolled up. The second manifold is called *S*-curve, because of its *S*-shape. Their parametric equations are given in (1.5) and (1.6) respectively. Both of them are developable surfaces, which can be flattened onto a plane without distortion. In mathematics, a developable surface is a surface with zero Gaussian curvature.

- Parametric equation of the Swiss roll:

$$\begin{cases} x = \left(\frac{3\pi}{2}(1+2t)\right) \cos\left(\frac{3\pi}{2}(1+2t)\right), \\ y = s, \\ z = \left(\frac{3\pi}{2}(1+2t)\right) \sin\left(\frac{3\pi}{2}(1+2t)\right), \end{cases} \quad 0 \leq s \leq L, |t| \leq 1. \quad (1.5)$$

- Parametric equation of the *S*-curve:

$$\begin{cases} x = -\cos(1.5\pi t), \\ y = s, \\ z = \begin{cases} -\sin(1.5\pi t), & 0 \leq t \leq 1, \\ 2 + \sin(1.5\pi t), & 1 < t \leq 2, \end{cases} \end{cases} \quad 0 \leq s \leq L, \quad 0 \leq t \leq 2. \quad (1.6)$$

Their graphs are shown in Fig. 1.6. An example of non-developable surface is

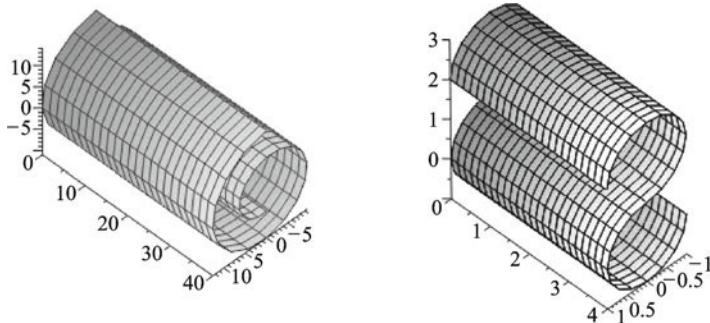


Fig. 1.6 Two benchmark manifold. Left: Swiss roll. Right: *S*-curve.

a punched sphere, which is obtained by cutting the top part of a sphere. Its parametric equation is the following, and its graph is on the left of Fig. 1.7.

$$\begin{cases} x = \cos(s) \sin(t), \\ y = \sin(s) \sin(t), \\ z = \cos(t). \end{cases} \quad 0 \leq s \leq 2\pi, \quad 0 \leq t \leq \pi, \quad (1.7)$$

The data set called 3D-cluster is also often used in the illustration of DR methods. The 3D-cluster is not a surface in the usual sense, but consists of

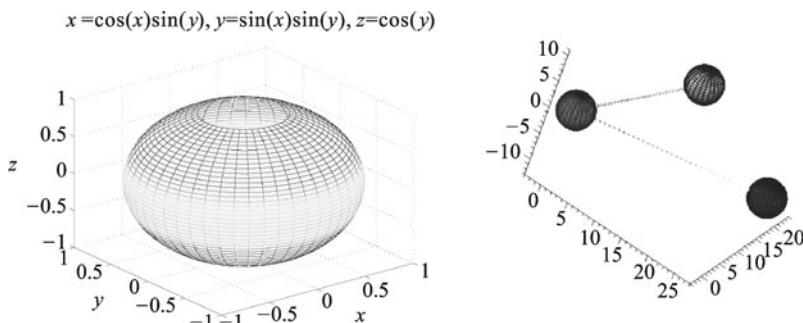


Fig. 1.7 Left: Punched sphere. Right: 3D-cluster.

three separated spheres, connected by two line segments, as shown on the right in Fig. 1.7.

References

- [1] Ashbaugh, D.R.: Ridgeology. *Journal of Forensic Identification* 41(1) (1991). URL <http://onin.com/fp/ridgeology.pdf>. Accessed 1 December 2011.
- [2] McCallum, A.K.: Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering (1996). <http://www.cs.cmu.edu/~mccallum/bow>. Accessed 1 December 2011.
- [3] Porter, M.: An algorithm for suffix stripping. *Program* 14(3), 130–137 (1980).
- [4] Bellman, R.: Adaptive Control Processes: A Guided Tour. Princeton University Press, Princeton (1961).
- [5] Scott, D.W., Thompson, J.R.: Probability density estimation in higher dimensions. In: J.E. Gentle (ed.) *Computer Science and Statistics: Proceedings of the Fifteenth Symposium on the Interface*, pp. 173–179. North Holland-Elsevier Science Publishers, Amsterdam, New York, Oxford (1983).
- [6] Posse, C.: Tools for two-dimensional exploratory projection pursuit. *Journal of Computational and Graphical Statistics* 4, 83–100 (1995).
- [7] Lee, J.A., Verleysen, M.: *Nonlinear Dimensionality Reduction*. Springer (2007).
- [8] Carreira-Perpiñán, M.Á.: A review of dimension reduction techniques. Tech. Rep. CS–96–09, Dept. of Computer Science, University of Sheffield, UK (1996).
- [9] Demartines, P.: Analyse de données par réseaux de neurones auto-organisés. Ph.D. thesis, Institut National Polytechnique de Grenoble, Grenoble, France (1994).
- [10] Aggarwal, C.C., Hinneburg, A., Kein, D.A.: On the surprising behavior of distance metrics in high dimensional space. In: J.V. den Busche, V. Vianu (eds.) *Proceedings of the Eighth International Conference on Database Theory, Lecture Notes in Computer Science*, vol. 1973, pp. 420–434. Springer, London (2001).
- [11] Beyer, K.S., Goldstein, J., U. Shaft, R.R.: When is “nearest neighbor” meaningful? In: *Proceedings of the Seventh International Conference on Database Theory, Lecture Notes in Computer Science*, vol. 1540, pp. 217–235. Springer-Verlag, Jerusalem, Israel (1999).
- [12] Borodin, A., Ostrovsky, R., Rabani, Y.: Low bounds for high dimensional nearest neighbor search and related problems. In: *Proceedings of the thirty-first annual ACM symposium on Theory of Computing*, Atlanta, GA, pp. 312–321. ACM Press, New York (1999).
- [13] Francois, D.: High-dimensional data analysis: optional metrics and feature selection. Ph.D. thesis, Université catholique de Louvain, Département d’Ingénierie Mathématique, Louvain-la-Neuve, Belgium (2006).
- [14] Erlebach, T., Jansen, K., Seidel, E.: Polynomial-time approximation schemes for geometric graphs. In: *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pp. 671–679 (2001).
- [15] Hastad, J.: Clique is hard to approximate within $n^{1-\varepsilon}$. In: *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pp. 627–636 (1996).

- [16] Pettis, K.W., Bailey, T.A., Jain, A.K., Dubes, R.C.: An intrinsic dimensionality estimator from near-neighbor information. *IEEE Trans on PAMI* 1, 25–37 (1979).
- [17] Verveer, P., Duin, R.: An evaluation of intrinsic dimensionality estimators. *IEEE Trans. on PAMI* 17(1), 81–86 (1995).
- [18] Kegl, B.: Intrinsic dimension estimation using packing numbers. in *Advanced NIPS* 14 (2002).
- [19] Little, A., Lee, J., Jung, Y.M., Maggioni, M.: Estimation of intrinsic dimensionality of samples from noisy low-dimensional manifolds in high dimensions with multiscale svd. In: *Statistical Signal Processing, IEEE/SP 15th Workshop*, pp. 85–88 (2009).
- [20] Lee, J.: Multiscale estimation of intrinsic dimensionality of point cloud data and multiscale analysis of dynamic graphs (2010).
- [21] Hein, M., Audibert, J.Y.: Intrinsic dimensionality estimation of submanifolds in r^d . In: *Proceedings of the 22nd international conference on Machine learning, ACM International Conference Proceeding Series*, vol. 119. ACM, New York (2005).
- [22] Costa, J., Hero, A.O.: Geodesic entropic graphs for dimension and entropy estimation in manifold learning. *IEEE Trans. on Signal Processing* 52, 2210–2221 (2004).
- [23] Levina, E., Bickel, P.: Maximum likelihood estimation of intrinsic dimension. In: L.B. L. K. Saul Y. Weiss (ed.) *Advances in NIPS* 17 (2005).
- [24] Tenenbaum, J.B., de Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. *Science* 290(5500), 2319–2323 (2000).
- [25] Weinberger, K.Q., Packer, B.D., Saul, L.K.: Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. In: *Proc. of the 10th International Workshop on AI and Statistics* (2005).
- [26] Gashler, M., Ventura, D., Martinez, T.: Iterative non-linear dimensionality reduction with manifold sculpting. In: J. Platt, D. Koller, Y. Singer, S. Roweis (eds.) *Advances in Neural Information Processing Systems* 20, pp. 513–520. MIT Press, Cambridge (2008).
- [27] Venna, J., Kaski, S.: Local multidimensional scaling. *Neural Networks* 19(6), 889–899 (2006).
- [28] Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. *Science* 290(5500), 2323–2326 (2000).
- [29] Zhang, Z.Y., Zha, H.Y.: Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. *SIAM J. Sci. Comput.* 26(1), 313–338 (2004).
- [30] Belkin, M., Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation* 15(6), 1373–1396 (2003).
- [31] Donoho, D.L., Grimes, C.: Hessian eigenmaps: New locally linear embedding techniques for high-dimensional data. *Proc. Natl. Acad. Sci. USA* 100, 5591–5596 (2003).
- [32] Coifman, R.R., Lafon, S.: Diffusion maps. *Appl. Comput. Harmon. Anal.* 21, 5–30 (2006).
- [33] Ultsch, A.: Emergence in self-organizing feature maps. In: *Workshop on Self-Organizing Maps (WSOM 07)*. Bielefeld (2007).
- [34] Bishop, C.M., Hinton, G.E., Strachan, I.G.D.: GTM through time. In: *IEE Fifth International Conference on Artificial Neural Networks*, pp. 111–116 (1997).

- [35] Bishop, C.M., Svensén, M., Williams, C.K.I.: GTM: The generative topographic mapping. *Neural Computation* 10(1), 215–234 (1998).
- [36] Candes, E., Tao, T.: Near optimal signal recovery from random projections: Universal encoding strategies? *IEEE Trans. on Information Theory* 52(12), 5406–5425 (2006).
- [37] Candes, E., Wakin, M.: An introduction to compressive sampling. *IEEE Trans. Signal Processing* 25(2), 21–30 (2008).
- [38] Donoho, D.: Compressed sensing. *IEEE Trans. on Information Theory* 52(4), 1289–1306 (2006).
- [39] Hayes, B.: The best bits. *American Scientist* 97(4), pp. 276 (2009).
- [40] Sarvotham, S., Baron, D., Baraniuk, R.: Measurements vs. bits: Compressed sensing meets information theory. In: Proc. Allerton Conference on Communication, Control, and Computing. Monticello, IL (2006).

Part I

Data Geometry

Chapter 2 Preliminary Calculus on Manifolds

Abstract This chapter provides a preliminary knowledge of manifold. Manifold geometry is the foundation of the geometric approach to dimensionality reduction. In DR, we assume that the observed (high-dimensional) data resides on a low-dimensional manifold whose dimension and shape are not a priori known. A manifold can be represented by its coordinates. DR methods try to find the coordinate representations of data. The current research of differential geometry focuses on the characterization of global properties of manifolds. However, DR methods only need the local properties of manifolds, which is much simpler. In this chapter, we give the notions and notations of manifolds. In Section 1, a linear manifold, also called a hyperplane, is introduced. In Section 2, we discuss differentiable structures of manifolds. An introduction to functions and operators on manifolds is given in Section 3. Most of the material in this chapter can be found in any standard textbook on differential geometry, for example, [1–4]. Therefore, the proofs of most theorems in this chapter are omitted.

2.1 Linear Manifold

The simplest manifold is a linear manifold, often called a hyperplane. A linear manifold is a local enlargement of a nonlinear manifold. Indeed, at each point of a nonlinear manifold, there exists a tangent space, which locally approximates the manifold. The tangent space is a linear manifold. Linear DR methods are based on the assumption that the observed data set resides on a linear manifold. In the study of a linear manifold, *Linear Algebra* is the main tool. Hence, we first briefly review some notions and notations in Linear Algebra.

The coordinates of a point \mathbf{x} in the Euclidean space \mathbb{R}^D are denoted by $\mathbf{x} = [x_1 \cdots x_D]'$. A point is also called a vector. Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ be a finite point set. It is often represented in the matrix form $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_n] = [x_{ij}]_{i,j=1}^{D,n}$. The data set \mathcal{X} spans a subspace of \mathbb{R}^D denoted by $S = \text{span}\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, which is the column space of the data matrix \mathbf{X} .

We denote the column space of a matrix \mathbf{M} by $\text{Col}(\mathbf{M})$ and the row space of \mathbf{M} by $\text{Row}(\mathbf{M})$. Hence, $\text{span}\{\mathbf{x}_1, \dots, \mathbf{x}_n\} = \text{Col}(\mathbf{X})$.

Remark 2.1. In this book, the notation x_j denotes the j th component of the vector \mathbf{x} , and the notation \mathbf{x}_j denotes the j th vector in the data set \mathcal{X} , whose i th component is $x_{i,j}$. The readers need to distinguish \mathbf{x}_j from \mathbf{x}_j carefully.

The Euclidean distance between two points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^D$ is denoted by $d_2(\mathbf{x}, \mathbf{y})$ or $\|\mathbf{x} - \mathbf{y}\|$, and the inner product of \mathbf{x} and \mathbf{y} is denoted by $\langle \mathbf{x}, \mathbf{y} \rangle$ or $\mathbf{x} \cdot \mathbf{y}$.

We use the notations $\mathfrak{M}_{m,n}$ and $\mathfrak{D}_{m,n}$ for the collections of all $m \times n$ matrices and its sub-collection of the diagonal matrices, respectively. In addition, the collection of all $n \times k$ matrices with orthonormal (o.n.) columns (if $k \leq n$), or with o.n. rows (if $n \leq k$), is denoted by $\mathfrak{O}_{n,k}$. For $k = n$, we replace $\mathfrak{M}_{m,n}$, $\mathfrak{D}_{n,k}$, and $\mathfrak{O}_{n,k}$ by \mathfrak{M}_m , \mathfrak{D}_n , and \mathfrak{O}_n , respectively. Particularly, \mathfrak{O}_n is the collection of $n \times n$ orthogonal (o.g.) matrices. We shall also denote the collection of all $n \times n$ symmetric matrices by \mathfrak{S}_n , the collection of all $n \times n$ (real-valued) positive semi-definite (psd) matrices by $S\mathfrak{P}_n$, and the collection of all positive definite (pd) matrices by \mathfrak{P}_n . If the rank of a matrix, say, $\mathbf{M} \in \mathfrak{M}_n$, is r and the rank needs to be emphasized, then the notation $\mathbf{M} \in \mathfrak{M}_n(r)$ will be used. The transpose of a matrix \mathbf{A} is denoted by either \mathbf{A}' or \mathbf{A}^T . Using the transpose notation, the inner product $\mathbf{x} \cdot \mathbf{y}$ can also be written as $\mathbf{x}'\mathbf{y}$ (or $\mathbf{x}^T\mathbf{y}$).

2.1.1 Subspace and Projection

Let $\mathcal{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_k\}$ be an o.n. basis of a k -dimensional subspace $S \subset \mathbb{R}^m$. Write $\mathbf{B} = [\mathbf{b}_1 \dots \mathbf{b}_k] \in \mathfrak{O}_{m,k}$. The orthogonal (o.g.) projection $\mathbf{P} : \mathbb{R}^m \rightarrow S$ can be represented by

$$\mathbf{y} = \mathbf{P}(\mathbf{x}) = \mathbf{B}\mathbf{B}'\mathbf{x}, \quad \mathbf{x} \in \mathbb{R}^m.$$

For a vector $\mathbf{y} \in S$,

$$\mathbf{y} = \mathbf{P}(\mathbf{y}) = \mathbf{B}\mathbf{B}'\mathbf{y} = \mathbf{B}\mathbf{y}_{[\mathcal{B}]}$$

where $\mathbf{y}_{[\mathcal{B}]} \stackrel{\text{def}}{=} \mathbf{B}'\mathbf{y}$ is called the \mathcal{B} -coordinates of \mathbf{y} . Since the dimension of S is k , S is isometrically isomorphic to \mathbb{R}^k , and the matrix \mathbf{B} defines an orthogonal embedding from \mathbb{R}^k to \mathbb{R}^m by

$$\mathbf{y} = \mathbf{B}\mathbf{z}, \quad \mathbf{z} \in \mathbb{R}^k.$$

We have

$$d_2(\mathbf{B}\mathbf{z}_1, \mathbf{B}\mathbf{z}_2) = d_2(\mathbf{z}_1, \mathbf{z}_2), \quad \mathbf{z}_1, \mathbf{z}_2 \in \mathbb{R}^k,$$

where the distance on the left-hand side is measured in \mathbb{R}^m while the distance on the right-hand side is measured in \mathbb{R}^k . Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^m$ and

$S = \text{span}\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$. Then the o.g. projection \mathbf{P} has the form

$$\mathbf{y} = \mathbf{P}(\mathbf{x}) = \mathbf{X}\mathbf{X}^{-}\mathbf{x},$$

where \mathbf{X}^{-} denotes a generalized inverse of \mathbf{X} (see Chapter 8 of [5]). When the vector set \mathcal{X} is linearly dependent, the generalized inverse \mathbf{X}^{-} is not unique. Otherwise, \mathbf{X}^{-} is unique and equal to the left inverse of \mathbf{X} , which can be represented by $\mathbf{X}^{-} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$.

To find an o.n. basis of $S = \text{Col}(\mathbf{X})$, we make the \mathbf{QR} factorization of \mathbf{X} :

$$\mathbf{X} = \mathbf{Q}\mathbf{R}, \quad (2.1)$$

where $\mathbf{Q} = [\mathbf{u}_1 \ \cdots \ \mathbf{u}_k] \in \mathfrak{O}_{m,k}$ and \mathbf{R} is a $k \times n$ upper triangular matrix. The set $\{\mathbf{u}_1, \dots, \mathbf{u}_k\}$ is an o.n. basis of S . Therefore, $\mathbf{Q}\mathbf{Q}'$ is the o.g. projection from \mathbb{R}^m to S . We claim that a generalized inverse of \mathbf{X} can be obtained by

$$\mathbf{X}^{-} = (\mathbf{R}'\mathbf{R})^{-1}\mathbf{X}'.$$

In fact, by $\mathbf{X}'\mathbf{X} = \mathbf{R}'\mathbf{R}$, we have,

$$\mathbf{X}\mathbf{X}^{-}\mathbf{X} = \mathbf{X}[(\mathbf{R}'\mathbf{R})^{-1}\mathbf{X}']\mathbf{X} = \mathbf{X},$$

which indicates that \mathbf{X}^{-} is a generalized inverse of \mathbf{X} .

Let $\mathbf{a} \in \mathbb{R}^m$ be given. The inner product $\mathbf{a} \cdot \mathbf{x}$ defines a linear function $L_{\mathbf{a}}(\mathbf{x}) = \mathbf{a} \cdot \mathbf{x}$ on \mathbb{R}^m , which is also called a linear functional on \mathbb{R}^m . The space of all linear functionals on a linear space S is called the dual space (or simply called dual) of S and denoted by S^* . We have $(\mathbb{R}^m)^* = \mathbb{R}^m$ and $\text{Col}(\mathbf{X})^* = \text{Row}(\mathbf{X})$. We have already shown that $\mathbf{X}\mathbf{X}^{-}$ is an o.g. projection from \mathbb{R}^m to $S = \text{Col}(\mathbf{X})$. By the similar discussion, we can show that $\mathbf{X}^{-}\mathbf{X}$ is an o.g. projection from \mathbb{R}^n to $S^* = \text{Row}(\mathbf{X})$. Let the \mathbf{QR} decomposition of \mathbf{X}' be

$$\mathbf{X}' = \mathbf{V}\hat{\mathbf{R}}, \quad (2.2)$$

where $\mathbf{V} = [\mathbf{v}_1 \ \cdots \ \mathbf{v}_k] \in \mathfrak{O}_{n,k}$. Then the column vector set $\mathcal{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ is an o.n. basis of S^* , and the o.g. projection $\mathbb{R}^n \rightarrow \text{Row}(\mathbf{X})$ can also be realized by $\mathbf{V}\mathbf{V}' = \mathbf{X}^{-}\mathbf{X}$.

A hyperplane is a shifted subspace. Let $S \subset \mathbb{R}^m$ be a k -dimensional subspace and $\mathbf{a} \in \mathbb{R}^m$. Then the shift of S through \mathbf{a} is the hyperplane

$$H = \mathbf{a} + S = \{\mathbf{x} + \mathbf{a} : \mathbf{x} \in S\}.$$

On the other hand, each hyperplane $H \subset \mathbb{R}^m$ has a unique parallel subspace S , which can be constructed as follows. Choose a vector $\mathbf{a} \in H$, then

$$S = H - \mathbf{a} = \{\mathbf{x} - \mathbf{a} : \mathbf{x} \in H\}.$$

The dimension of a hyperplane is equal to the dimension of its parallel subspace.

For a real number $c \in \mathbb{R} \setminus \{0\}$, the c -dilation of the date set $\mathcal{X} \subset \mathbb{R}^m$ is defined by $c\mathcal{X} = \{cx : x \in \mathcal{X}\}$, and, for an o.g. matrix $\mathbf{O} \in \mathfrak{O}_m$, the rotation of \mathcal{X} is set as $\hat{\mathcal{X}} = \{\mathbf{O}x_1, \dots, \mathbf{O}x_n\}$. It is clear that shift, dilation and rotation do not change the dimension of a subspace.

When a hyperplane H is not a subspace, we often represent a point $x \in H$ by

$$\mathbf{x} = \mathbf{a} + \sum_{i=1}^k \mathbf{b}_i \langle \mathbf{x}, \mathbf{b}_i \rangle,$$

where $\mathbf{a} \in H$ and $\{\mathbf{b}_1, \dots, \mathbf{b}_k\}$ is an o.n. basis of the parallel subspace of H .

2.1.2 Functions on Euclidean Spaces

We already know that each vector $\mathbf{a} \in \mathbb{R}^m$ defines a linear functional by

$$L_{\mathbf{a}}(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle, \quad \mathbf{x} \in \mathbb{R}^m, \tag{2.3}$$

which is a *homogeneous linear function* on \mathbb{R}^m . Later, we shall identify the term *homogeneous linear function* with *linear functional*. The constant function $L_c(\mathbf{x}) = c$, $c \in \mathbb{R}$, is an inhomogeneous linear function.

Let $\{\mathbf{e}_1, \dots, \mathbf{e}_m\}$ be the canonical o.n. basis of \mathbb{R}^m (also called the coordinate basis). Then each \mathbf{e}_i defines a functional with $L_{\mathbf{e}_i}(\mathbf{x}) = x_i$, which is often denoted by the δ -notation $\delta_i : \delta_i(\mathbf{x}) = L_{\mathbf{e}_i}(\mathbf{x})$, and called the i th coordinate function. The set $\{1, \mathbf{e}_1, \dots, \mathbf{e}_m\}$ is a basis of the space of linear functions on \mathbb{R}^m . Let $S = \text{span } \mathcal{X}$. We can find o.n. bases of S and S^* using the QR decomposition of \mathbf{X} and \mathbf{X}' in (2.1) and (2.2) respectively. They can also be obtained from the singular value decomposition of \mathbf{X} :

$$\mathbf{X} = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^T,$$

where $\{\mathbf{u}_1, \dots, \mathbf{u}_k\}$ is an o.n. basis of S and $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ is an o.n. basis of S^* . The duality of S and S^* is very useful in many applications, because S^* characterizes S and vice versa.

To deal with nonlinear functions on \mathbb{R}^m , we introduce the following notations. We denote the set of natural integers by \mathbb{N} , the set of nonnegative integers by $\mathbb{Z}_+ = \mathbb{N} \cup \{0\}$, and the set of nonnegative real numbers by \mathbb{R}_+ . For $\mathbf{n} \in \mathbb{Z}_+^m$, we define $\mathbf{n}! = n_1! \cdots n_m!$, $|\mathbf{n}| = n_1 + \cdots + n_m$, and for $\mathbf{x} \in \mathbb{R}^m$, we define $\mathbf{x}^\mathbf{n} = x_1^{n_1} \cdots x_m^{n_m}$. The m -multivariate Gamma function is denoted by $\Gamma(\mathbf{x}) = \Gamma(x_1) \cdots \Gamma(x_m)$. For $i, j \in \{1, 2, \dots, m\}$ and $s \in \mathbb{Z}_+$, we write $D_i = \frac{\partial}{\partial x_i}$, $D_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j}$, and $D_i^s = \frac{\partial^s}{\partial x_i^s}$. For $\mathbf{k} \in \mathbb{Z}_+^m$, we write

$$D^{\mathbf{k}} = D_1^{k_1} \cdots D_m^{k_m} = \frac{\partial^{|\mathbf{k}|}}{\partial x_1^{k_1} \cdots \partial x_m^{k_m}}.$$

$$D^{\mathbf{k}} = D_1^{k_1} \cdots D_m^{k_m} = \frac{\partial^{|\mathbf{k}|}}{\partial x_1^{k_1} \cdots \partial x_m^{k_m}}.$$

The gradient operator is denoted by either ∇ or $\mathbf{D} \stackrel{\text{def}}{=} [D_1, \dots, D_m]$, and a linear differential operator $v_1 D_1 + \dots + v_m D_m$ is denoted by $\mathbf{D}\mathbf{v}$ or $\nabla\mathbf{v}$. A non-vanished linear operator is a (non-normalized) directional derivative operator in the direction \mathbf{v} . Note that we represent the gradient operator \mathbf{D} as a row-vector. Hence, $\mathbf{D}\mathbf{v}$ can be considered as the matrix product of \mathbf{D} and \mathbf{v} . We have

$$(\mathbf{D}\mathbf{v})^2 = (\mathbf{D}\mathbf{v})(\mathbf{D}\mathbf{v}) = \left(\sum_{i=1}^m v_i D_i \right)^2 = \sum_{i=1}^m \sum_{j=1}^m v_i v_j D_{i,j}.$$

Let \mathbf{H} be the matrix

$$\mathbf{H} = \begin{pmatrix} D_{1,1} & \cdots & D_{1,m} \\ D_{2,1} & \cdots & D_{2,m} \\ \vdots & & \vdots \\ D_{m,1} & \cdots & D_{m,m} \end{pmatrix}. \quad (2.4)$$

Then $(\mathbf{D}\mathbf{v})^2 = \mathbf{v}^T \mathbf{H} \mathbf{v}$. By the mathematical induction, we define $(\mathbf{D}\mathbf{v})^k = (\mathbf{D}\mathbf{v})^{k-1}(\mathbf{D}\mathbf{v})$.

We assume that the readers have already learned the Lebesgue spaces $L^p(\mathbb{R}^m)$, $1 \leq p \leq \infty$, and the spaces of continuously differentiable functions, $C^k(\mathbb{R}^m)$, $k \in \mathbb{Z}_+ \cup \{\infty\}$. Based on them, the Sobolev spaces $H^k(\mathbb{R}^m)$, $k \in \mathbb{Z}^+$, is defined by

$$\mathbf{H}^k(\mathbb{R}^m) = \{f : \mathbf{D}^j f \in L^2(\mathbb{R}^m), j \in \mathbb{Z}_+^m, |j| \leq k\}.$$

Let $U \subset \mathbb{R}^m$ be an open set. Then the spaces $L^p(U)$, $C^k(U)$, and $H^k(U)$ can be defined in the similar way. A vector-valued function $f = [f^1, \dots, f^s]' : \Omega \subset \mathbb{R}^m \rightarrow \mathbb{R}^s$ is usually called a mapping. We write $f \in \mathbf{H}^k(U)$ if $f^j \in H^k(U)$, $1 \leq j \leq s$.

Let $B_{\mathbf{x}_0}(r) \subset \mathbb{R}^m$ be an open ball centered at \mathbf{x}_0 and $f \in C^{k+1}(B_{\mathbf{x}_0}(r))$. The Taylor's formula for $f(\mathbf{x})$, $\mathbf{x} \in B_{\mathbf{x}_0}(r)$, is

$$f(\mathbf{x}) = \sum_{j=1}^k \frac{1}{j!} (\mathbf{D}\Delta\mathbf{x})^j f(\mathbf{x}_0) + \frac{1}{(k+1)!} (\mathbf{D}\Delta\mathbf{x})^{k+1} f(\mathbf{x}_0 + \theta\Delta\mathbf{x}), \quad 0 < \theta < 1,$$

where $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}_0$. We rewrite the quadratic term in the Taylor's expansion as

$$\frac{1}{2} (\mathbf{D}\Delta\mathbf{x})^2 = \frac{1}{2} \Delta\mathbf{x}' \mathbf{H}(f)(\mathbf{x}_0) \Delta\mathbf{x},$$

where $\mathbf{H}(f)(\mathbf{x}_0) = [D_{i,j} f(\mathbf{x}_0)]_{i,j=1}^m$ is called the Hessian matrix of f at \mathbf{x}_0 . Sometimes, we also write $\mathbf{H}_f(\mathbf{x}_0) = \mathbf{H}(f)(\mathbf{x}_0)$.

2.1.3 Laplace Operator and Heat Diffusion Kernel

Let $\Omega \subset \mathbb{R}^m$ be an open bounded domain. The Laplace operator on $H^2(\Omega)$ is defined by

$$\Delta = \nabla \cdot \nabla = \frac{\partial^2}{\partial x_1^2} + \cdots + \frac{\partial^2}{\partial x_m^2}, \quad (2.5)$$

and the Laplacian eigenvalue problem on the closed domain $\bar{\Omega} = \Omega \cup \partial\Omega$ is the equation of the form

$$Lu \stackrel{\text{def}}{=} -\Delta u = \lambda u \quad \text{in } \Omega, \quad (2.6)$$

with one of the following boundary conditions (BCs):

$$\begin{cases} u = 0, & \text{on } \partial\Omega \text{ (Dirichlet BC),} \\ \frac{\partial u}{\partial \vec{\nu}} = 0, & \text{on } \partial\Omega \text{ (Neumann BC),} \\ \frac{\partial u}{\partial \vec{\nu}} + au = 0, & \text{on } \partial\Omega \text{ (Robin BC),} \end{cases} \quad (2.7)$$

where $\vec{\nu}$ is the normal direction of $\partial\Omega$ and $a \in \mathbb{R}$. If a function $u \neq 0$ satisfies the Laplacian eigenvalue problem with one of the above BCs, then we say that u is an *eigenfunction* achieving (or corresponding to) the *eigenvalue* λ . Usually, we normalize the eigenfunction u by $\|u\| = 1$.

In data analysis, we only consider either Dirichlet-Laplacian problem or Neumann-Laplacian problem. We shall simply call either of them a Laplacian eigenvalue problem if the BC need not to be emphasized.

It is known that $L = -\Delta$ is a self-adjoint operator. Hence, the set of eigenvalues of (2.6) is countable, and each eigenvalue has a finite multiplicity. We order them increasingly,

$$0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_k \leq \cdots \longrightarrow \infty.$$

If Dirichelet BC is applied, $\lambda_1 > 0$, else if Neumann BC applied, $\lambda_1 = 0$. Let φ_k , $k = 1, 2, \dots$, be the corresponding eigenfunctions:

$$L\varphi_k = \lambda_k \varphi_k.$$

They form an o.n. basis of $L^2(\Omega)$ so that each $f \in L^2(\Omega)$ has the *eigenfunction expansion*:

$$f = \sum_{k=1}^{\infty} \langle f, \varphi_k \rangle \varphi_k. \quad (2.8)$$

In mathematical literature, people also often call L the Laplace operator, or Laplacian. In this book, we adopt this term. Laplacian has a close relation with the heat diffusion operator. For simplicity, let us consider the heat

equation with Dirichlet BC on an open bounded domain Ω :

$$\begin{cases} u_t = \Delta u & \text{in } (0, \infty) \times \Omega, \\ u = 0 & \text{on } (0, \infty) \times \partial\Omega, \\ u(\mathbf{x}, 0) = u_0(\mathbf{x}) & \text{in } \Omega. \end{cases} \quad (2.9)$$

By the exponential formula, the Green's function $\rho_t(\mathbf{x}, \mathbf{y})$ (also called the kernel) for (2.9) is

$$\rho_t(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{\infty} e^{-\lambda_j t} \varphi_j(\mathbf{x}) \overline{\varphi_j(\mathbf{y})}, \quad t \in (0, \infty), \quad (\mathbf{x}, \mathbf{y}) \in \overline{\Omega} \times \overline{\Omega},$$

which yields the solution of (2.9):

$$u(\mathbf{x}, t) = (e^{t\Delta} u_0)(\mathbf{x}) = \sum_{j=1}^{\infty} e^{-t\lambda_j} \langle u_0, \varphi_j \rangle \varphi_j(\mathbf{x}).$$

The diffusion operator $e^{t\Delta}$ has the same eigenfunction set as the Laplacian L , and the range of its eigenvalues is in $[0, 1]$. If the Neumann BC is applied, then the largest eigenvalue of $e^{t\Delta}$ is 1.

When $\Omega = \mathbb{R}^m$, we set the following Cauchy initial problem:

$$\begin{cases} u_t = \Delta u, & \text{in } (0, \infty) \times \mathbb{R}^m, \\ u(\mathbf{x}, 0) = u_0(\mathbf{x}), & \text{in } \mathbb{R}^m, \end{cases} \quad (2.10)$$

where $u_0(\mathbf{x})$ is a bounded continuous function and the solution $u(\mathbf{x}, t)$ is also required to be bounded. The Green's function for (2.10) is the Gaussian kernel:

$$G_t(\mathbf{x}, \mathbf{y}) = \frac{1}{(2\sqrt{\pi t})^m} e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{4t}}, \quad (2.11)$$

which yields the solution of (2.10):

$$u(\mathbf{x}, t) = \int_{\mathbb{R}^m} G_t(\mathbf{x}, \mathbf{y}) u_0(\mathbf{y}) d\mathbf{y}.$$

2.2 Differentiable Manifolds

2.2.1 Coordinate Systems and Parameterization

Differentiable manifold is a mathematical concept, which generalizes smooth curves and surfaces to n -dimensional geometric objects. In order to give a

formal definition of manifold, we need the concept of *differentiable homeomorphism*. Let $U \subset \mathbb{R}^k$ and $V \subset \mathbb{R}^l$ be two open sets. A mapping $f : U \rightarrow V$ is called smooth if all partial derivatives $\frac{\partial^s f}{\partial x_{i_1} \cdots \partial x_{i_s}}$, $s \in \mathbb{Z}$, exist and are continuous (i.e., $f \in C^\infty$). In general, assume that $X \subset \mathbb{R}^k$ and $Y \subset \mathbb{R}^l$ are two arbitrary non-empty sets. A mapping $f : X \rightarrow Y$ is called a smooth mapping if, for each $x \in X$, there is an open set $U \subset \mathbb{R}^k$ with $x \in U$ and a smooth mapping $F : U \rightarrow \mathbb{R}^l$ such that $F = f$ on $X \cap U$. For convenience, we shall simply call $X \cap U$ the open set on X . A mapping is called bijection if it is one-to-one and onto.

Definition 2.1. Let $X \subset \mathbb{R}^k$ and $Y \subset \mathbb{R}^l$ be two arbitrary non-empty sets. If a bijection $f : X \rightarrow Y$ and its inverse f^{-1} both are smooth, then f is called a differentiable homeomorphism (or diffeomorphism), and X is said to be diffeomorphic to Y .

Roughly speaking, a differentiable manifold is a geometric object that is locally diffeomorphic to an open set of a Euclidean space. The following is its formal definition.

Definition 2.2. Let $M \subset \mathbb{R}^m$ be non-empty. If, for each point $x \in M$, there is an open set $W \subset M$ such that W is diffeomorphic to an open set $U \subset \mathbb{R}^k$, then M is called a k -dimensional differentiable manifold (or smooth manifold), a diffeomorphism $g : U \rightarrow W$ is called a parameterization of W , its inverse $h (= g^{-1}) : W \rightarrow U$ called a *coordinate mapping*, W called a coordinate neighborhood, and the couple (W, h) called a (local) coordinate system or a chart on M .

In a coordinate system (W, h) , a point $x \in W$ has the coordinates

$$h(x) = [h^1(x), \dots, h^m(x)],$$

where h^i is called the i th coordinate function.

Later, a k -dimensional differentiable manifold will be simply called a k -manifold. By Definition 2.2, a manifold is “glued” by local coordinate neighborhoods. Integrating them together leads to a structure on the manifold.

Definition 2.3. A differentiable structure (or an atlas) \mathcal{D} on a k -manifold M is a collection of all coordinate systems $\{(W_i, h_i)\}$ on M , satisfying the following conditions.

- The union of W_i covers M : $M \subset \cup_i W_i$.
- For each pair (i, j) , $h_j \circ h_i^{-1}$ is a smooth mapping on $h_i(W_i \cap W_j)$.

In topology and related branches of mathematics, a Hausdorff space is defined as a topological space in which distinct points have disjoint neighborhoods. A k -manifold therefore can be considered as a Hausdorff space, which is locally homeomorphic to \mathbb{R}^k and has a countable basis. We give an example to demonstrate the differentiable structure on a manifold.

Example 2.1. The unit spherical surface $\mathbb{S}^2 \subset \mathbb{R}^3$ has the Cartesian equation

$$\mathbb{S}^2 : x^2 + y^2 + z^2 = 1.$$

Let $W_1 = \left\{ (x, y, z) \in \mathbb{S}^2 : z > -\frac{1}{2} \right\}$ and $W_2 = \left\{ (x, y, z) \in \mathbb{S}^2 : z < \frac{1}{2} \right\}$, and define

$$g_1 : \begin{cases} x = \frac{2u}{u^2 + v^2 + 1}, \\ y = \frac{2v}{u^2 + v^2 + 1}, \\ z = \frac{1 - u^2 - v^2}{u^2 + v^2 + 1}. \end{cases}$$

Then g_1 is a parameterization of W_1 , mapping $U = u^2 + v^2 < 3$ to W_1 , and

$$h_1 : \begin{cases} u = \frac{x}{1+z}, \\ v = \frac{y}{1+z}, \end{cases}$$

is the coordinate mapping.

Similarly, the parameterization of W_2 ,

$$g_2 : \begin{cases} x = \frac{2\tilde{u}}{\tilde{u}^2 + \tilde{v}^2 + 1}, \\ y = \frac{2\tilde{v}}{\tilde{u}^2 + \tilde{v}^2 + 1}, \\ z = \frac{\tilde{u}^2 + \tilde{v}^2 - 1}{\tilde{u}^2 + \tilde{v}^2 + 1}, \end{cases} \quad (2.12)$$

maps U to W_2 , and the coordinate mapping from W_2 to U is

$$h_2 : \begin{cases} \tilde{u} = \frac{x}{1-z}, \\ \tilde{v} = \frac{y}{1-z}. \end{cases}$$

The collection $\{(W_1, h_1), (W_2, h_2)\}$ is a differentiable structure on \mathbb{S}^2 . For a point $\mathbf{p} \in W_1 \cap W_2$, Let $g_1(u, v) = \mathbf{p}$ and $g_2(\tilde{u}, \tilde{v}) = \mathbf{p}$. Then the following mapping f , from the open set $h_2(W_2) \subset \mathbb{R}^2$ to the open set $h_1(W_1) \subset \mathbb{R}^2$, is a diffeomorphism:

$$f : \begin{cases} u = \frac{\tilde{u}}{\tilde{u}^2 + \tilde{v}^2}, \\ v = \frac{\tilde{v}}{\tilde{u}^2 + \tilde{v}^2}. \end{cases} \quad (2.13)$$

Note that the spherical parametric equation $g : [0, 2\pi] \times [0, \pi] \rightarrow \mathbb{S}^2$ defined by

$$g : \begin{cases} x = \cos \varphi \sin \phi, \\ y = \sin \varphi \sin \psi, \\ z = \cos \psi, \end{cases}$$

is not a diffeomorphism on the whole manifold.

As a generalization of the concept of subspace, submanifold is defined as follows.

Definition 2.4. Let M be a d -manifold and \mathcal{D} be a differentiable structure on M . A subset $S \subset M$ is called a k -submanifold of M if, for each coordinate system $(W, h) \in \mathcal{D}$, $h(S \cap W)$ is a diffeomorphism of an open set $O \subset \mathbb{R}^k$.

With the aid of differentiable structure, we are able to define the differentiable (or, smooth) function on any open set $V \subset M$.

Definition 2.5. Let \mathcal{D} be a differentiable structure on M . A function f on an open set $V \subset M$ is called differentiable (or, smooth) if, for each coordinate system $(W, h) \in \mathcal{D}$, $f \circ h^{-1}$ is differentiable on $h(V \cap W)$.

We denote the class of differentiable functions on $V \subset M$ by $\mathfrak{F}(V)$. Let $f, g \in \mathfrak{F}(V)$ and $c \in \mathbb{R}$. Then cf , $f + g$, and fg are in $\mathfrak{F}(V)$. It can be verified that the differentiability of a function on V does not rely on the choice of particularly coordinate systems on M .

If there is a single coordinate system (M, h) on a manifold M such that each point $\mathbf{x} \in M$ has the coordinates $h(\mathbf{x})$, then we call M a *simple manifold*. High-dimensional data sets in the real-world applications usually can be modeled as subsets of simple manifolds. Hence, we assume that the underlying manifold of a high-dimensional data set is simple, unless a contrary claim is given.

2.2.2 Tangent Spaces and Tangent Vectors

Let $M \subset \mathbb{R}^m$ be a manifold. Passing through a point $\mathbf{p} \in M$, we can construct the hyperplane H that “best” approximates M in a neighborhood of \mathbf{p} . Intuitively, the tangent space $T_{\mathbf{p}}$ is the subspace parallel to the hyperplane H . For an open set $U \subset \mathbb{R}^k$, its tangent space $T_{\mathbf{x}}U$, $\mathbf{x} \in U$, is trivially defined as the whole space \mathbb{R}^k . To define tangent space for a nonlinear manifold, we first recall the concept of the derivative of a differentiable mapping $f = [f_1, \dots, f_l]' : U \rightarrow V$, where U and V are open sets of \mathbb{R}^k and \mathbb{R}^l respectively. At $\mathbf{x} \in U$, the derivative of f is the linear mapping $df_{\mathbf{x}} : \mathbb{R}^k \rightarrow \mathbb{R}^l$ defined by

$$df_{\mathbf{x}} = \lim_{t \rightarrow 0} \frac{f(\mathbf{x} + t\mathbf{h}) - f(\mathbf{x})}{t}, \quad \mathbf{h} \in \mathbb{R}^k, \quad (2.14)$$

which can be represented by the matrix

$$df_{\mathbf{x}} = \left[\frac{\partial f_i(\mathbf{x})}{\partial x_j} \right]_{i,j=1}^{l,k} = \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial f_1(\mathbf{x})}{\partial x_k} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial f_2(\mathbf{x})}{\partial x_k} \\ \vdots & & \vdots \\ \frac{\partial f_l(\mathbf{x})}{\partial x_1} & \dots & \frac{\partial f_l(\mathbf{x})}{\partial x_k} \end{pmatrix}. \quad (2.15)$$

Particularly, when $l = 1$, f is reduced to a function $f : U \rightarrow \mathbb{R}$, and the derivative of f is identified with the gradient of f :

$$df_{\mathbf{x}} (= Df_{\mathbf{x}}) = \nabla f_{\mathbf{x}} = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \dots, \frac{\partial f(\mathbf{x})}{\partial x_k} \right].$$

Derivatives of smooth mappings have the following properties.

Lemma 2.1. 1. If $f : U \rightarrow V$ and $g : V \rightarrow W$ both are smooth mappings and $f(\mathbf{x}) = \mathbf{y}$, then

$$d(g \circ f)_{\mathbf{x}} = dg_{\mathbf{y}} df_{\mathbf{x}}.$$

2. If $U \subset \tilde{U}$ and $i : U \rightarrow \tilde{U}$ is the inclusion mapping, i.e., $i(\mathbf{x}) = \mathbf{x}$ for each $\mathbf{x} \in U$, then $di_{\mathbf{x}}$ is the identity mapping on \mathbb{R}^k .
3. If $L : \mathbb{R}^k \rightarrow \mathbb{R}^l$ is a linear transformation, represented by the matrix \mathbf{L} such that $L(\mathbf{x}) = \mathbf{L}\mathbf{x}$, $\mathbf{x} \in \mathbb{R}^k$, then the derivative of L at each \mathbf{x} is \mathbf{L} : $dL_{\mathbf{x}} = \mathbf{L}$.

By the properties of derivatives, it is easy to verify that if $U \subset \mathbb{R}^k$, $V \subset \mathbb{R}^l$, and $f : U \rightarrow V$ is a diffeomorphism, then $k = l$ and $df_{\mathbf{x}}$ is non-degenerate, i.e., the matrix $df_{\mathbf{x}}$ is invertible. The result is partially reversible according to the following well-known Inverse Function Theorem.

Theorem 2.1. If the derivative $df_{\mathbf{x}}$ is non-degenerate, then there is a neighborhood U around \mathbf{x} such that f is a diffeomorphism from U to $f(U)$.

We now define the tangent space of a manifold.

Definition 2.6. Let $M \subset \mathbb{R}^m$ be a k -manifold, $U \subset \mathbb{R}^k$ be an open set, and $g : U \rightarrow M$ be a parameterization of the neighborhood $g(U) \subset M$. Assume $\mathbf{p} \in M$, $\mathbf{u} \in U$, and $g(\mathbf{u}) = \mathbf{p}$. Then the image of the linear transformation $dg_{\mathbf{u}}$, denoted by $T_{\mathbf{p}}M \stackrel{\text{def}}{=} dg_{\mathbf{u}}(\mathbb{R}^k)$, is called the tangent space of M (at \mathbf{p}). A vector in $T_{\mathbf{p}}M$ is called a tangent vector.

The tangent space $T_{\mathbf{p}}M$ is shortly denoted by $T_{\mathbf{p}}$ if M need not be emphasized. Because $dg_{\mathbf{u}}$ is a full-rank matrix, $T_{\mathbf{p}}M$ is a k -dimensional subspace of \mathbb{R}^m .

Remark 2.1. Sometimes, for convenience, we also call the hyperplane $H_{\mathbf{p}} \stackrel{\text{def}}{=} \mathbf{p} + T_{\mathbf{p}}$ the tangent space of M (through \mathbf{p}), for $H_{\mathbf{p}}$ is exactly tangent to M .

We write

$$\frac{\partial g}{\partial u^i} = \left[\frac{\partial g_1}{\partial u^i}, \dots, \frac{\partial g_m}{\partial u^i} \right]^T, \quad 1 \leq i \leq k.$$

By Definition 2.6, the set

$$\left\{ \frac{\partial g}{\partial u^1}, \dots, \frac{\partial g}{\partial u^k} \right\} \quad (2.16)$$

is a basis of $T_p M$. Define the basis matrix by

$$\frac{\partial g}{\partial u} \stackrel{\text{def}}{=} \left[\frac{\partial g_i}{\partial u^j} \right]_{i,j=1}^{m,k} = \begin{pmatrix} \frac{\partial g_1}{\partial u^1} & \cdots & \frac{\partial g_1}{\partial u^k} \\ \frac{\partial g_2}{\partial u^1} & \cdots & \frac{\partial g_2}{\partial u^k} \\ \vdots & & \vdots \\ \frac{\partial g_m}{\partial u^1} & \cdots & \frac{\partial g_m}{\partial u^k} \end{pmatrix}. \quad (2.17)$$

Then a tangent vector at p has the form

$$\mathbf{X}_p = \sum_{i=1}^k \alpha_i \frac{\partial g}{\partial u^i} = \frac{\partial g}{\partial u} \vec{\alpha}, \quad (2.18)$$

where $\vec{\alpha} = [\alpha_1, \dots, \alpha_k]' \in \mathbb{R}^k$. If $\vec{\alpha} \neq 0$, it represents a direction in \mathbb{R}^k and \mathbf{X}_p is a direction in the tangent space $T_p M$. If we choose an isometric mapping g , then the matrix $\frac{\partial g}{\partial u}$ has orthonormal columns, so that $\|\mathbf{X}_p\| = \|\vec{\alpha}\|$.

Assume that f is a function on M , which has a smooth extension on an open set O of \mathbb{R}^m such that $M \subset O$. Then we can compute the derivative of f as a function on $O \subset \mathbb{R}^m$. Composed of g , it builds a composite function $f \circ g$ on \mathbb{R}^k . On the other hand, as a function on M , f can also be represented as a function of the coordinates u , say $F(u)$. These two functions are identical: $f \circ g(u) = F(u)$. Then the directional derivatives of F in the direction $\vec{\alpha} \in \mathbb{R}^k$ can be computed by

$$\frac{\partial f \circ g}{\partial \alpha} = df_p \frac{\partial g}{\partial u} \vec{\alpha}. \quad (2.19)$$

We now consider the coordinate change on tangent space. Assume that $g_1 : U \rightarrow M$ and $g_2 : V \rightarrow M$ are two parameterizations so that $g_1(\mathbf{u}) = p$ and $g_2(\mathbf{v}) = p$. The tangent space of M at p can be represented by each of them. The parameterization g_1 generates a basis of $T_p M$ through the basis matrix

$$\frac{\partial g_1}{\partial u} \stackrel{\text{def}}{=} \left[\frac{\partial g_{1,i}}{\partial u^j} \right]_{i,j=1}^{m,k}$$

and the parameterization g_2 generates a basis of $T_p M$ through the basis matrix

$$\frac{\partial g_2}{\partial v} \stackrel{\text{def}}{=} \left[\frac{\partial g_{2,i}}{\partial v^j} \right]_{i,j=1}^{m,k}.$$

Let \mathbf{X}_p be a tangent vector on $T_p M$ such that

$$\mathbf{X}_p = \sum_{i=1}^k \alpha_i \frac{\partial g_1}{\partial u^i} = \frac{\partial g_1}{\partial u} \vec{\alpha}.$$

Assume under the basis of $\left\{ \frac{\partial g_2}{\partial v^j} \right\}_{j=1}^k$, it is

$$\mathbf{X}_p = \sum_{i=1}^k \beta_i \frac{\partial g_2}{\partial v^i} = \frac{\partial g_2}{\partial v} \vec{\beta}.$$

We then have

$$\vec{\beta} = \left(\frac{\partial g_2}{\partial v} \right)^{-} \frac{\partial g_1}{\partial u} \vec{\alpha},$$

where $\left(\frac{\partial g_2}{\partial v} \right)^{-}$ is the left-inverse of $\frac{\partial g_2}{\partial v}$. Corresponding to g_1 and g_2 , let the coordinate mappings be h_1 and h_2 respectively. Then the mapping $h_2 \circ g_1 : U \rightarrow V$ is a diffeomorphism and $h_2 \circ g_1(\mathbf{u}) = \mathbf{v}$. Meanwhile, the linear mapping $d(h_2 \circ g_1)_{\mathbf{u}}$ is a non-degenerate linear transformation on \mathbb{R}^k . By Lemma 2.1, we have

$$d(h_2 \circ g_1)_{\mathbf{u}} = (dh_2)_{\mathbf{p}}(dg_1)_{\mathbf{u}}.$$

Since $h_2 = g_2^{-1}$,

$$(dh_2)_{\mathbf{p}} = \left(\frac{\partial g_2}{\partial v} \right)^{-}.$$

We have already proved the following.

Theorem 2.2. *Let g_1 and g_2 be two parameterizations of M and $T_p M$ be the tangent space of M at p . Let $\frac{\partial g_1}{\partial u}$ and $\frac{\partial g_2}{\partial v}$ be the basis matrices of $T_p M$ corresponding to g_1 and g_2 respectively. Let \mathbf{X}_p be a tangent vector having the coordinates $\vec{\alpha}$ associated with $\frac{\partial g_1}{\partial u}$ and the coordinates $\vec{\beta}$ associated with $\frac{\partial g_2}{\partial v}$. Then*

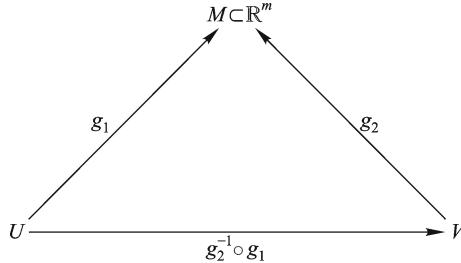
$$\vec{\beta} = \frac{\partial v}{\partial u} \vec{\alpha}, \tag{2.20}$$

$$\frac{\partial g_2}{\partial v} = \frac{\partial g_1}{\partial u} \frac{\partial u}{\partial v}, \tag{2.21}$$

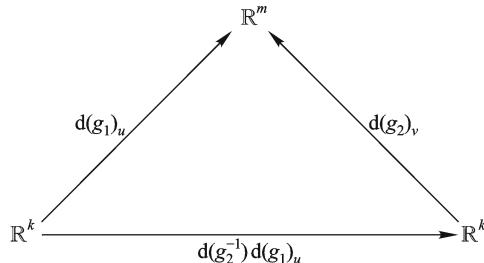
where

$$\frac{\partial v}{\partial u} = d(g_2^{-1} \circ g_1)_{\mathbf{u}}, \quad \frac{\partial u}{\partial v} = d(g_1^{-1} \circ g_2)_{\mathbf{v}}. \tag{2.22}$$

The relations between the smooth mappings on open sets U and V are illustrated in the following diagram.



The relation between smooth mappings above induces the relation between the following linear mappings.



Theorem 2.2 shows the importance of tangent space in local analysis of functions on manifold: When we want to deal with a function on manifold, we use its derivative and work on tangent space. We verify Theorem 2.2 by the following example.

Example 2.2. We show by steps the coordinate change between different bases for the tangent space of \mathbb{S}^2 . Let (W_1, g_1) , (W_2, g_2) , and f be defined as in Example 2.1. Let $\mathbf{p} = [x(u, v), y(u, v), z(u, v)]' \in W_1$. By the definition, the derivative $(dg_1)_{(u,v)}$ is the matrix

$$\mathbf{D}g_{1(u,v)} = \frac{1}{(u^2 + v^2 + 1)^2} \begin{bmatrix} 2(v^2 - u^2 + 1) & -4uv \\ -4uv & 2(u^2 - v^2 + 1) \\ -4u & -4v \end{bmatrix}. \quad (2.23)$$

The column vectors of $\mathbf{D}g_{1,p}$, written by \mathbf{t}_1 and \mathbf{t}_2 , are a basis of the tangent space of \mathbb{S}^2 at \mathbf{p} . If $\mathbf{p} \in W_2$ too, then $\mathbf{p} = [\tilde{x}(\tilde{u}, \tilde{v}), \tilde{y}(\tilde{u}, \tilde{v}), \tilde{z}(\tilde{u}, \tilde{v})]'$ has the explicit formula given in (2.12). We have

$$\mathbf{D}g_{2(\tilde{u},\tilde{v})} = \frac{1}{(\tilde{u}^2 + \tilde{v}^2 + 1)^2} \begin{bmatrix} 2(\tilde{v}^2 - \tilde{u}^2 + 1) & -4\tilde{u}\tilde{v} \\ -4\tilde{u}\tilde{v} & 2(\tilde{u}^2 - \tilde{v}^2 + 1) \\ 4\tilde{u} & 4\tilde{v} \end{bmatrix}. \quad (2.24)$$

The column vectors $\tilde{\mathbf{t}}_1$ and $\tilde{\mathbf{t}}_2$ also form a basis of the tangent space of \mathbb{S}^2 at \mathbf{p} . We verify that the coordinate change is given by df . Indeed, by (2.13), the linear mapping $df_{(\tilde{u}, \tilde{v})}$ on \mathbb{R}^2 is given by the matrix

$$df_{(\tilde{u}, \tilde{v})} = \frac{1}{(\tilde{u}^2 + \tilde{v}^2)^2} \begin{bmatrix} \tilde{v}^2 - \tilde{u}^2 & -2\tilde{u}\tilde{v} \\ -2\tilde{u}\tilde{v} & \tilde{u}^2 - \tilde{v}^2 \end{bmatrix} = \begin{bmatrix} v^2 - u^2 & -2uv \\ -2uv & u^2 - v^2 \end{bmatrix}, \quad (2.25)$$

where the last equality is obtained from (2.13). The direct computation shows

$$[\tilde{\mathbf{t}}_1 \ \tilde{\mathbf{t}}_2] = [\mathbf{t}_1 \ \mathbf{t}_2] df_{(\tilde{u}, \tilde{v})}, \quad (2.26)$$

which verifies (2.21).

2.2.3 Riemannian Metrics

A manifold will become a metric space when it is endowed with a metric. We do not intend to introduce Riemannian metrics in a general sense, limiting the content to what we shall use in the book. Let M be a k -manifold and $T_{\mathbf{p}}$ the tangent space at $\mathbf{p} \in M$. Assume that g is a parameterization of M and (W, h) is the coordinate system on M , where $h = g^{-1}$. For each $\mathbf{p} \in M$, let $\mathbf{u} = h(\mathbf{p}) \in \mathbb{R}^k$, then g defines a basis (2.16) on $T_{\mathbf{p}}M$, which can be represented by the basis matrix $\frac{\partial g}{\partial u}$ in (2.17). The metric $\mathbf{G}_g(\mathbf{p})$ is defined by

$$\mathbf{G}_g(\mathbf{p}) = \left(\frac{\partial g}{\partial u} \right)^T \frac{\partial g}{\partial u}, \quad \mathbf{p} \in M. \quad (2.27)$$

The $\mathbf{G}_g(\mathbf{p})$ is a pd matrix, called a Riemannian metric on M . Sometimes we simply denote it by $\mathbf{G}(\mathbf{p})$ or \mathbf{G} . Let two tangent vectors $\mathbf{X}, \mathbf{Y} \in T_{\mathbf{p}}M$ be represented by $\mathbf{X} = \frac{\partial g}{\partial u} \mathbf{x}$ and $\mathbf{Y} = \frac{\partial g}{\partial u} \mathbf{y}$ respectively. Their inner product is defined by

$$\langle \mathbf{X}, \mathbf{Y} \rangle_{\mathbf{p}} = \mathbf{x}^T \mathbf{G}(\mathbf{p}) \mathbf{y}. \quad (2.28)$$

It is easy to verify the following.

Lemma 2.2. *For any tangent vectors $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \in T_{\mathbf{p}}$ and real numbers $a, b \in \mathbb{R}$, the following properties hold.*

Linearity: $\langle a\mathbf{X} + b\mathbf{Y}, \mathbf{Z} \rangle_{\mathbf{p}} = a\langle \mathbf{X}, \mathbf{Z} \rangle_{\mathbf{p}} + b\langle \mathbf{Y}, \mathbf{Z} \rangle_{\mathbf{p}}$,

Symmetry: $\langle \mathbf{X}, \mathbf{Y} \rangle_{\mathbf{p}} = \langle \mathbf{Y}, \mathbf{X} \rangle_{\mathbf{p}}$,

Positive definiteness: $\langle \mathbf{X}, \mathbf{X} \rangle_{\mathbf{p}} \geq 0$, and the equality holds if and only if $\mathbf{X} = \mathbf{0}$.

The inner product defines the norm of \mathbf{X} by

$$\|\mathbf{X}\|_{\mathbf{p}}^2 = \mathbf{x}^T \mathbf{G}(\mathbf{p}) \mathbf{x}. \quad (2.29)$$

Riemannian manifold is defined in the following.

Definition 2.7. Let M be a k -manifold. If M is endowed with the metric $\mathbf{G}_g(\mathbf{p})$, and for each pair of tangent vectors, the inner product is defined by (2.28), then M is called a Riemannian manifold and denoted by (M, g) or (M, \mathbf{G}) .

In this book, a Riemannian metric will be simply called a metric and a Riemannian manifold will be simply called a manifold. We point out that the defined inner product is independent of the selection of the metric, i.e., independent of the parameterizations of M .

Theorem 2.3. Let g_1 and g_2 be two parameterizations of M and $\mathbf{T}_{\mathbf{p}}M$ be the tangent space of M at \mathbf{p} . Let \mathbf{X} and \mathbf{Y} be two tangent vectors on $\mathbf{T}_{\mathbf{p}}M$ such that under g_1 and g_2 , their coordinates are \mathbf{x}_1 , \mathbf{y}_1 , and \mathbf{x}_2 , \mathbf{y}_2 , respectively. Then

$$\mathbf{x}_1^T \mathbf{G}_{g_1}(\mathbf{p}) \mathbf{y}_1 = \mathbf{x}_2^T \mathbf{G}_{g_2}(\mathbf{p}) \mathbf{y}_2, \quad (2.30)$$

and the metric change formula is

$$\mathbf{G}_{g_2}(\mathbf{p}) = \left(\frac{\partial u}{\partial v} \right)_{\mathbf{p}}^T \mathbf{G}_{g_1}(\mathbf{p}) \left(\frac{\partial u}{\partial v} \right)_{\mathbf{p}}. \quad (2.31)$$

Proof. By Theorem 2.2, we have

$$\frac{\partial g_2}{\partial v} = \frac{\partial g_1}{\partial u} \left(\frac{\partial u}{\partial v} \right),$$

which yields

$$\begin{aligned} \mathbf{G}_{g_2}(\mathbf{p}) &= \left(\frac{\partial g_2}{\partial v} \right)^T \frac{\partial g_2}{\partial v} \\ &= \left(\frac{\partial u}{\partial v} \right)_{\mathbf{p}}^T \left(\frac{\partial g_1}{\partial u} \right)^T \frac{\partial g_1}{\partial u} \left(\frac{\partial u}{\partial v} \right)_{\mathbf{p}} \\ &= \left(\frac{\partial u}{\partial v} \right)_{\mathbf{p}}^T \mathbf{G}_{g_1}(\mathbf{p}) \left(\frac{\partial u}{\partial v} \right)_{\mathbf{p}}. \end{aligned}$$

The equation (2.31) is proved. By Theorem 2.2, we also have

$$\begin{aligned} \mathbf{x}_1 &= \left(\frac{\partial u}{\partial v} \right)_{\mathbf{p}} \mathbf{x}_2, \\ \mathbf{y}_1 &= \left(\frac{\partial u}{\partial v} \right)_{\mathbf{p}} \mathbf{y}_2. \end{aligned}$$

Hence,

$$\begin{aligned} \mathbf{x}_2^T \mathbf{G}_{g_2}(\mathbf{p}) \mathbf{y}_2 &= \mathbf{x}_2^T \left(\frac{\partial u}{\partial v} \right)_{\mathbf{p}}^T \mathbf{G}_{g_1}(\mathbf{p}) \left(\frac{\partial u}{\partial v} \right)_{\mathbf{p}} \mathbf{y}_2 \\ &= \mathbf{x}_1^T \mathbf{G}_{g_1}(\mathbf{p}) \mathbf{y}_1. \end{aligned}$$

The theorem is proved. □

In many applications, we choose the parameterization g to be isometric so that $\mathbf{G}_g(\mathbf{p}) = \mathbf{I}$. In these cases, manifold metric is similar to Euclidean metric. Recall that on Euclidean space, the identity mapping is a parameterization, which generates the Identity Euclidean metric. The following example illustrates Riemannian metrics on \mathbb{S}^2 .

Example 2.3. Recall that (W_1, h_1) and (W_2, h_2) are two local coordinate systems on the spherical surface \mathbb{S}^2 (see Example 2.1). Let $\mathbf{p} = (x, y, z) \in W_1$, and $(u, v) = h_1(x, y, z)$. Then g_1 and g_2 in Example 2.1 define two Riemannian metrics on \mathbb{S}^2 . We now compute the metrics $\mathbf{G}_{g_1}(\mathbf{p})$ and $\mathbf{G}_{g_2}(\mathbf{p})$ respectively. Let the vectors $\{\mathbf{t}_1, \mathbf{t}_2\}$ be defined in (2.23). We have

$$\begin{aligned}\mathbf{G}_{g_1}(\mathbf{p}) &= \begin{bmatrix} \langle \mathbf{t}_1, \mathbf{t}_1 \rangle & \langle \mathbf{t}_1, \mathbf{t}_2 \rangle \\ \langle \mathbf{t}_2, \mathbf{t}_1 \rangle & \langle \mathbf{t}_2, \mathbf{t}_2 \rangle \end{bmatrix} \\ &= \frac{4}{(u^2 + v^2 + 1)^2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.\end{aligned}$$

Similarly, assume that $\tilde{\mathbf{p}} = (\tilde{x}, \tilde{y}, \tilde{z}) \in W_2$ and $(\tilde{u}, \tilde{v}) = h_2(\tilde{x}, \tilde{y}, \tilde{z})$. By (2.28), we have

$$\mathbf{G}_{g_2}(\mathbf{p}) = (\mathbf{D}_{g_2}(\tilde{u}, \tilde{v}))^\top \mathbf{D}_{g_2}(\tilde{u}, \tilde{v}) = \frac{4}{(\tilde{u}^2 + \tilde{v}^2 + 1)^2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

If $\tilde{\mathbf{p}} = \mathbf{p} \in W_1 \cap W_2$, then the two matrices above satisfy the relation in (2.31). Recall that the linear transformation from $\{\mathbf{t}_1, \mathbf{t}_2\}$ to $\{\tilde{\mathbf{t}}_1, \tilde{\mathbf{t}}_2\}$ is $\mathrm{d}f_{(\tilde{u}, \tilde{v})}$ in (2.25). A simple computation yields

$$(\mathrm{d}f_{(\tilde{u}, \tilde{v})})^\top \mathbf{G}_{g_1}(\mathbf{p}) \mathrm{d}f_{(\tilde{u}, \tilde{v})} = \frac{4(u^2 + v^2)^2}{(u^2 + v^2 + 1)^2} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

By (2.13), we have

$$\frac{4(u^2 + v^2)^2}{(u^2 + v^2 + 1)^2} = \frac{4}{(\tilde{u}^2 + \tilde{v}^2 + 1)^2},$$

which yields

$$\mathbf{G}_{g_2}(\mathbf{p}) = (\mathrm{d}f_{(\tilde{u}, \tilde{v})})^\top \mathbf{G}_{g_1}(\mathbf{p}) \mathrm{d}f_{(\tilde{u}, \tilde{v})}.$$

2.2.4 Geodesic Distance

The arc length on a manifold M is defined in the following.

Definition 2.8. Let M be a connected Riemannian manifold and $\gamma \subset M$ is a curve on M with the parametric equation $\gamma = [\gamma^1(t), \dots, \gamma^m(t)]'$, $t \in [a, b]$. Then the length of the curve γ is defined by

$$\|\gamma\| = \int_a^b \|\dot{\gamma}\| dt = \int_a^b \sqrt{\sum_i^m (\dot{\gamma}^i)^2} dt, \quad (2.32)$$

where $\dot{\gamma}$ is the derivative of γ with respect to t .

The arc length is independent of the choice of the parametric equation of the curve. Particularly, we can choose the arc length as the parameter of a curve. A curve is parameterized by the arc length if and only if $\|\dot{\gamma}(t)\| = 1$ for all $t \in [a, b]$.

Let \mathbf{p} and \mathbf{q} be two points in M . The geodesic distance between them is defined by

$$d_g(\mathbf{p}, \mathbf{q}) = \inf\{\|\gamma\|\},$$

where the infimum extends over all smooth curves connecting \mathbf{p} and \mathbf{q} .

2.3 Functions and Operators on Manifolds

2.3.1 Functions on Manifolds

Let $M \subset \mathbb{R}^m$ be a k -manifold, g be a parameterization of M and $\mathbf{u} = [u^1, \dots, u^k]$ be the corresponding coordinate system on M . Then there is a domain $U \subset \mathbb{R}^k$ such that $h(M) = U$. For convenience, we identify u with a point in U . We already define smooth functions on M in Definition 2.5. With the help of coordinates on M , we can represent a function f on M as a function on the Euclidean domain U : $f : U \rightarrow \mathbb{R}$, $f(u) = f(\mathbf{x}(u))$, where $\mathbf{x} \in M$ with $u(\mathbf{x}) = u \in \mathbb{R}^k$. Since the domain of f now is Euclidean, the discussion in Subsection 2.1.2 for functions on Euclidean space is also valid for f , as a function on the manifold M . For example, we have Taylor's formula of $f(u)$ around a point $u_0 \in U$ with $g(u_0) = \mathbf{x}_0$:

$$f(u) = \sum_{j=1}^k \frac{1}{j!} (D\Delta u)^j f(u_0) + \frac{1}{(k+1)!} (D\Delta u)^{k+1} f(u_0 + \theta\Delta u), \quad 0 < \theta < 1,$$

where $\Delta u = u - u_0$.

We shall call f a linear function (under the coordinates u) if $f(u) = c + \sum_{i=1}^k a_i u^i = c + u\mathbf{a}$, where $\mathbf{a} = [a_1, \dots, a_k]' \in \mathbb{R}^k$, and call f quadratic if

$$f(u) = c + \sum_{i=1}^k a_i u^i + \sum_{i,j=1}^k b_{ij} u_i u_j = c + u\mathbf{a} + u\mathbf{B}u',$$

where $\mathbf{B} = [b_{ij}]$ is a $k \times k$ matrix.

The i th partial derivative of f is denoted by $\frac{\partial f}{\partial u^i}$, or simply $\partial_i f$ if the coordinates are not to be emphasized. In many applications, the extension of f on an open set $O \subset \mathbb{R}^m$, $M \subset O$, is known, then we can compute ∂f by (2.19). Let $g(\mathbf{x}) = u$. We have

$$\partial_i f = Df_{\mathbf{x}} \frac{\partial g}{\partial u^i} = \sum_{j=1}^m \frac{\partial f}{\partial x_j} \frac{\partial g_j}{\partial u^i}.$$

2.3.2 Operators on Manifolds

Let the Riemannian metric derived from g be denoted by $\mathbf{G} \stackrel{\text{def}}{=} \mathbf{G}_g$, which is a matrix-function on M . Since it is pd, $|\mathbf{G}| \stackrel{\text{def}}{=} \det(\mathbf{G}) > 0$ everywhere. For convenience, we write the inverse $\mathbf{G}^{-1} = [g^{ij}]$ (recalling that $\mathbf{G} = [g_{ij}]$). The gradient of f on M with respect to g is a vector field defined by

$$\text{grad } f = [(\text{grad } f)^1, \dots, (\text{grad } f)^k], \quad (2.33)$$

with

$$(\text{grad } f)^i = \sum_{j=1}^k g^{ij} \partial_j f.$$

The divergence of a vector field $\mathbf{X} = [X^1, \dots, X^k]$ on M is defined by

$$\text{div } \mathbf{X} = \frac{1}{\sqrt{|\mathbf{G}|}} \sum_{i=1}^k \partial_i \left(\sqrt{|\mathbf{G}|} X^i \right). \quad (2.34)$$

Definition 2.9. Laplace-Beltrami operator on $C^2(M)$ is defined by

$$\Delta f = \text{div grad } f = \sum_{i=1}^k \sum_{j=1}^k \frac{1}{\sqrt{|\mathbf{G}|}} \partial_i \left(\sqrt{|\mathbf{G}|} g^{ij} \partial_j f \right). \quad (2.35)$$

Note that we also often call $-\Delta$ Laplace-Beltrami operator as we deal with the Laplacian. All of these operators are dependent on coordinates. In DR, we often choose g to be isometric. Then $|\mathbf{G}| = 1$ and $g_{ij} = g^{ij} = \delta_{ij}$, so that their formulas are reduced to the simple expressions:

$$\text{grad } f = \left[\frac{\partial}{\partial u^1}, \dots, \frac{\partial}{\partial u^k} \right]$$

$$\text{div } \mathbf{X} = \sum_{i=1}^k \partial_i X^i$$

and

$$\Delta f = \sum_{i=1}^k \frac{\partial^2 f}{\partial(u^i)^2}.$$

Finally, we discuss the function spaces on M . In the following, we always assume that M is connected, bounded, and oriented. Let g be the parameterization of M , which generates the metric \mathbf{G} , and $g(M) = U \subset \mathbb{R}^k$. The volume of M is defined by

$$\text{Vol}(M) = \int_U \sqrt{|\mathbf{G}|} du. \quad (2.36)$$

Using the metric change formula (2.31), we can prove that the definition of volume in (2.36) is independent of the selection of coordinates. Hence, we adopt the coordinate-free notation to rewrite (2.36) as

$$\text{Vol}(M) = \int_M 1.$$

Remark 2.2. Assume that M has a single coordinate mapping h whose domain $U \subset \mathbb{R}^k$ is connected, bounded, and oriented. Assume also that h is smooth on the closed set $U \cup \partial U$. Then M is connected, bounded, and oriented.

In general, for a smooth function f on M , we define the integral of f over M by

$$\int_M f = \int_U f \sqrt{\mathbf{G}} du. \quad (2.37)$$

Since the integral is independent of the chosen coordinates, we again apply the coordinate-free notation on the right-hand side of (2.37).

A function f defined on M is called square integrable if $|f|^2$ is integrable. The space of all square integrable functions on M , denoted by $H(M)$, is a Hilbert space equipped with the inner product

$$\langle f, h \rangle = \int_M fh.$$

The following theorem can be derived from the Divergence Theorem [4].

Theorem 2.4. Let $\mathbf{X} = [X^1, \dots, X^k]^T$ be a C^1 vector field on M and f is a compactly supported function on M so that $f = 0$ on ∂M . Then the following equation holds.

$$\int_M \text{div } f \mathbf{X} = 0.$$

By Theorem 2.4, we obtain the following.

Theorem 2.5. Let f and q be compactly supported functions on M . Then

$$\int_M f \Delta q = \int_M q \Delta f, \quad (2.38)$$

$$\int_M \|\operatorname{grad} f\|^2 = - \int_M f \Delta f. \quad (2.39)$$

Proof. By $\operatorname{div} f \mathbf{X} = \langle \operatorname{grad} f, \mathbf{X} \rangle + f \operatorname{div} \mathbf{X}$ and Theorem 2.4, we obtain

$$\int_M \langle \operatorname{grad} f, \mathbf{X} \rangle = - \int_M f \operatorname{div} \mathbf{X}. \quad (2.40)$$

Particularly, choosing $\mathbf{X} = \operatorname{grad} q$ for a function $q \in C^2$ and applying (2.5), we get

$$- \int_M f \Delta q = \int_M \langle \operatorname{grad} f, \operatorname{grad} q \rangle, \quad (2.41)$$

which yields

$$\int_M f \Delta q = \int_M q \Delta f$$

and

$$\int_M \|\operatorname{grad} f\|^2 = - \int_M f \Delta f. \quad (2.42)$$

The proof is completed. \square

References

- [1] Boothby, W.M.: An Introduction to Differentiable Manifolds and Riemannian Geometry. Academic Press (1975).
- [2] Flanders, H.: Differential forms with applications to the physical sciences. Dover (1989).
- [3] Jost, J.: Riemannian Geometry and Geometric Analysis. Springer-Verlag (2002).
- [4] de Rham, G.: Differentiable Manifolds. Springer-Verlag (1984).
- [5] Rao, C., Rao, M.: Matrix Algebra and Its Applications to Statistics and Econometric. World Scientific, Singapore (1998).

Chapter 3 Geometric Structure of High-Dimensional Data

Abstract In applications, a high-dimensional data is given as a discrete set in a Euclidean space. If the points of data are well sampled on a manifold, then the data geometry is inherited from the manifold. Since the underlying manifold is hidden, it is hard to know its geometry by the classical manifold calculus. The data graph is a useful tool to reveal the data geometry. To construct a data graph, we first find the neighborhood system on the data, which is determined by the similarity (or dissimilarity) among the data points. The similarity information of data usually is driven by the application in which the data are used. In this chapter, we introduce the methods for defining the data similarity (or dissimilarity). We also introduce the preliminary spectral graph theory to analyze the data geometry. In Section 1, the construction of neighborhood system on data is discussed. The neighborhood system on a data set defines a data graph, which can be considered as a discrete form of a manifold. In Section 2, we introduce the basic concepts of graphs. In Section 3, the spectral graph analysis is introduced as a tool for analyzing the data geometry. Particularly, the Laplacian on a graph is briefly discussed in this section. Most of the materials in Sections 2 and 3 are found in [1–3].

3.1 Similarity and Dissimilarity of Data

Dimensionality reduction is based on the assumption that the observed high-dimensional data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ lies on a low-dimensional d -manifold $M \subset \mathbb{R}^D$, $d \ll D$. In many applications, the data sets are finite. Hence, we consider the data set \mathcal{X} as a set of sampling points on the manifold M . However, the data usually is not directly sampled from an existent manifold. For example, when we digitize a text document into a vector (see Subsection 1.2.3), the vector is built by counting the keywords, not sampled from a known manifold. Similarly, in face recognition, a vector in a data set may be an ordered gray-level values of pixels, and in the representation of a hyperspectral image, the vector is a discrete spectral curve at a raster cell of

the HSI cube. All of these data sets are not directly sampled from existent manifolds, but from digitized objects. Therefore, the similarities and/or dissimilarities between the objects of a data set are mathematically described by a *neighborhood* system.

3.1.1 Neighborhood Definition

Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ be a given data set and $\varepsilon > 0$ is a tolerance. The ε -neighborhood of a point $\mathbf{x}_i \in \mathcal{X}$ is a subset of \mathcal{X} defined by

$$O_i = O_i(\varepsilon) = \mathcal{X} \cap B_\varepsilon(i) \setminus \{\mathbf{x}_i\},$$

where $B_\varepsilon(i) \subset \mathbb{R}^D$ is the open sphere centered at \mathbf{x}_i of radius ε . It is clear that if $\mathbf{x}_j \in O_i$, then $\mathbf{x}_i \in O_j$. There is a practical difficulty to define ε -neighborhood on a data set. First, the value of the tolerance ε is dependent on the data dimension and scale. An object can be digitized in different scales. For example, an image can be digitized by either gray-level or intensity. If 8-bit scale (a byte) is used for recording the gray-level at a pixel, then it is in the integer range [0, 255]. However, if it is digitized by intensity, then its pixel values are in the range [0, 1]. The raw hyperspectral image data are often scaled within the integer range [0, 1000]. Hence, in many applications, \mathcal{X} and its dilation $c\mathcal{X}$ ($c > 0$) are considered as the same data set. Thus, the value of ε has to be adjusted according to the range of data. More seriously, as we pointed out in Section 1.3, when the data extrinsic dimension D is high, the points in the ε -neighborhood will concentrate around its boundary. Therefore, the ε becomes very unstable in accurately describing the data similarity. In DR, people often use k -neighborhood [4] to describe the data similarity. A k -neighborhood of \mathbf{x}_i contains the k nearest points of \mathbf{x}_i , excluding itself. Let $N(i) = N_k(i)$ denote the k -neighborhood of \mathbf{x}_i . Then,

$$\max_{\mathbf{x} \in N(i)} d_2(\mathbf{x}_i, \mathbf{x}) \leq \min_{\mathbf{x} \notin N(i) \cup \{\mathbf{x}_i\}} d_2(\mathbf{x}_i, \mathbf{x}).$$

Since a k -neighborhood is independent of data dimension and scale, in most cases, it is more suitable to describe data similarity than the ε -neighborhood. Note that $\mathbf{x}_j \in N(i)$ does not imply $\mathbf{x}_i \in N(j)$. We show this fact in Fig. 3.1. A k -neighborhood system sometimes cannot truly describe the geometric structure of a data set. For example, isolated points in a data set should have no neighbors. However, when we construct the k -neighborhood system on the data, each point has k neighbors ($k > 1$). It distorts the isolate property of the points. This type of problems is called neighborhood scaling problems [5], which will become critical in the application of anomaly detection. The authors of [5] suggest to construct hybrid neighborhood in such applications, utilizing the Principal Axis Tree (AP-Tree) technique. More discussion of the construction of hybrid neighborhood can be found in [6–12].

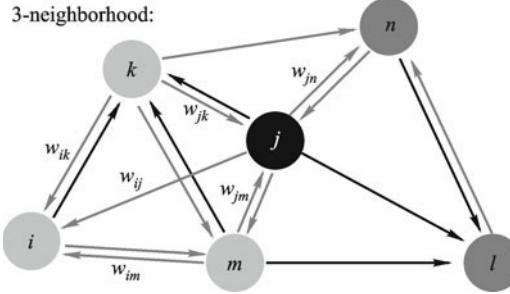


Fig. 3.1 In the figure, the 3-neighborhood system is constructed on a data set with 6 nodes. Each node is labeled by a letter and its nearest 3 neighbors are connected to it by arcs (arrows). The nodes labeled by k, m and j are the neighbors of i , while the nodes labeled by k, m and n are the neighbors of j . Hence, j is a neighbor of i , but i is not a neighbor of j .

Finally, we remark that although Euclidean distance is usually used in the construction of neighborhood, in many applications, other distances can also be adopted for various purposes. We list some of them in the following:

Angular distance

The angular distance between two (non-zero) vectors $\mathbf{x}, \mathbf{a} \in \mathbb{R}^D$ is defined by

$$d_{ang}(\mathbf{x}, \mathbf{a}) = \|\mathbf{x} - \mathbf{a}\|_a = \sqrt{\sum_{i=1}^D \left| \frac{x_i}{\|\mathbf{x}\|} - \frac{a_i}{\|\mathbf{a}\|} \right|^2}.$$

Let the angle between \mathbf{x} and \mathbf{a} be denoted by θ . If all components of these two vectors are non-negative so that $0 \leq \theta \leq \frac{\pi}{2}$, then

$$d_{ang}(\mathbf{x}, \mathbf{a}) = 2 \sin \frac{\theta}{2},$$

which is an increasing function of θ . Observe that $d_{ang}(\mathbf{x}, \mathbf{a}) \approx \theta$ for small values of θ . Hence, the angular distance approximately measures the angle between two non-negative vectors.

Template distance

In mathematics, template distance is also called l_∞ distance. If the peak term of a vector is one of its main characteristics, then the l_∞ distance between two vectors defined by

$$d_\infty(\mathbf{x}, \mathbf{a}) = \|\mathbf{x} - \mathbf{a}\|_\infty = \max_{1 \leq i \leq D} (|x_i - a_i|)$$

is a suitable measurement for dissimilarity.

Fourier distance

Since noise mostly occurs in the high frequency range of the Fourier domain, to reduce the impact of noise, we define the Fourier distance between two vectors $\mathbf{s}, \mathbf{a} \in \mathbb{R}^D$ by cutting off the higher frequency components. To compute the Fourier distance, we first use Fast Fourier Transform (FFT) to map each vector to the Fourier domain, then delete the desired high frequency terms. Thus, the Fourier distance is defined by

$$\hat{d}_2(\mathbf{x}, \mathbf{a}) = \sqrt{\sum_{i=1}^s (\text{FFT}(\mathbf{x} - \mathbf{a}))_i^2}, \quad 1 \leq s < D,$$

where $\text{FFT}(\mathbf{v})$ denotes the FFT of the vector \mathbf{v} , and s is some suitable integer smaller than D . Note that the sum only contains the lower frequency terms of the FFT in order to reduce noise. Of course, for $s = D$, we have $\hat{d}_2(\mathbf{x}, \mathbf{a}) = d_2(\mathbf{x}, \mathbf{a})$.

Wavelet distance

The wavelet transform (see, for example, [13, 14, 15]) is a useful tool in data analysis, particularly for separating noise from data content [16]. The discrete wavelet transform of a vector can be adopted to effectively distinguish its features from noise. To eliminate noise contamination, we suggest the following wavelet distance on the data set. Let the single-level discrete wavelet transform DWT of a vector $\mathbf{x} \in \mathbb{R}^D$ be denoted by

$$\text{DWT}(\mathbf{x}) = L(\mathbf{x}) + H(\mathbf{x}),$$

where $L(\mathbf{x})$ is in the low frequency wavelet subspace V_1 and $H(\mathbf{x})$ is in the high frequency wavelet subspace W_1 . Then the wavelet distance between \mathbf{x} and \mathbf{a} is defined by

$$d_w(\mathbf{x}, \mathbf{a}) = \|L(\mathbf{x}) - L(\mathbf{a})\|.$$

Modifications of this definition could include some high frequency terms of a multi-level DWT. The readers may refer to [17] and references therein for more detailed discussion of this discrete wavelet transform approach and its properties.

Fourier phase distance

The Fourier phase distance is used to measure the dissimilarity of two vectors \mathbf{x} and \mathbf{a} in terms of the phases of their FFT, defined by

$$\hat{d}_\theta(\mathbf{x}, \mathbf{a}) = \sqrt{\sum_{i=1}^s (\arg(\text{FFT}(\mathbf{x}))_i - \arg(\text{FFT}(\mathbf{a}))_i)^2}, \quad 1 \leq s < D,$$

which only takes the phase difference of the corresponding harmonics into consideration.

Derivative distance

This distance definition is a special case of distances in homogeneous Sobolev spaces. It is also called the edge distance for image data, since it measures the dissimilarity in terms of edges of two images of the same size. It is well known that edges play a very important role in describing image features. Hence, the edge distance well describes the dissimilarity on the data set whose elements are images. The derivative distance is formulated as

$$d_{der}(\mathbf{x}, \mathbf{a}) = \sqrt{\sum_{i=1}^{D-1} (\text{diff}(\mathbf{x} - \mathbf{a}))_i^2},$$

where “diff” denotes a desirable difference operator.

Fourier phase-difference distance

This distance definition adopts the same idea of the derivative distance, but the difference of Fourier phases is considered as follows.

$$\hat{d}_{d\theta}(\mathbf{x}, \mathbf{a}) = \sqrt{\sum_{i=1}^s (\text{diff}(\arg \text{FFT}(\mathbf{x}) - \arg \text{FFT}(\mathbf{a})))_i^2}, \quad 1 \leq s < D.$$

Directionally weighted distance

Let \mathbf{M} be a $D \times D$ positive definite (pd) matrix, which defines a metric on \mathbb{R}^D . Then the directionally weighted distance between two vectors \mathbf{a} and \mathbf{x} associated with \mathbf{M} is defined by

$$d_\mu(\mathbf{x}, \mathbf{a}) = (\mathbf{x} - \mathbf{a})' \mathbf{M}^{-1} (\mathbf{x} - \mathbf{a}). \quad (3.1)$$

The above weighted distance can be explained as follows. Suppose that $\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_D$, are the most important, linearly independent D vectors for a given data set, usually called *feature vectors* in *machine learning* [18]. Assume also that their importance are measured by a certain set of weights $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_D > 0$. Since the vectors $\{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_D\}$ constitute a basis of \mathbb{R}^D so that the matrix $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_D]$ is invertible, we may define a distance between two vectors \mathbf{a} and \mathbf{x} by

$$d_\lambda(\mathbf{x}, \mathbf{a}) = [\mathbf{B}^{-1}(\mathbf{x} - \mathbf{a})]' \mathbf{A} \mathbf{B}^{-1} (\mathbf{x} - \mathbf{a}), \quad (3.2)$$

to classify the vectors in the data set according to the importance of $\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_D$, where

$$\mathbf{A} = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_D).$$

Note that $\mathbf{B}^{-1}\mathbf{x}$ and $\mathbf{B}^{-1}\mathbf{a}$ are the \mathbf{B} -coordinates of the vectors \mathbf{x} and \mathbf{a} , respectively. Now, setting

$$\mathbf{M} = \mathbf{B}' \mathbf{A}^{-1} \mathbf{B},$$

we have

$$d_\lambda(\mathbf{x}, \mathbf{a}) = d_\mu(\mathbf{x}, \mathbf{a}).$$

We remark that this distance is an extension of the well known Mahalanobis distance, which is first introduced by Mahalanobis in [19] (also see [20]).

3.1.2 Algorithms for Construction of Neighborhood

In this subsection we develop the algorithms for constructing neighborhood system on a data set. In algorithm development, either a k -neighborhood or an ε -neighborhood on a data set is derived by an appropriate sorting procedure based on using the square Euclidean distance matrix on the data set. Assume that the given data set is $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$, which is corresponding to the data matrix $\mathbf{X} = [\mathbf{x}_1 \ \dots \ \mathbf{x}_n]$. The square Euclidean distance matrix on \mathcal{X} is defined by

$$\mathbf{S} = [d_2^2(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^n. \quad (3.3)$$

Write the matrix

$$\boldsymbol{\Phi} = \text{diag}(\|\mathbf{x}_1\|^2, \dots, \|\mathbf{x}_n\|^2) \mathbf{E},$$

where \mathbf{E} is the $n \times n$ matrix with the same value 1 for all of its entries. Then the matrix \mathbf{S} can be computed by the formula

$$\mathbf{S} = \boldsymbol{\Phi} + \boldsymbol{\Phi}' - 2\mathbf{X}'\mathbf{X}. \quad (3.4)$$

We use an $n \times n$ boolean matrix $\mathbf{A} = [e_{ij}]$ to represent the neighborhood structure of \mathcal{X} , where $e_{ij} = 1$ if \mathbf{x}_j is a neighbor of \mathbf{x}_i ; otherwise, $e_{ij} = 0$. If the ε -neighborhood is applied, then \mathbf{A} is symmetric. But it may be asymmetric about the k -neighborhood.

In the following, we first give Matlab code for the computation of the square Euclidean distance matrix \mathbf{S} in (3.3).

```
function S = square_distance(X)
% compute square Euclidean distance matrix for data.
% Syntax: S = SQUARE_DISTANCE(X)
% INPUT:
% X: (D x n) matrix representing n points in R^D.
% OUTPUT:
% S: (n x n) square Euclidean distances matrix.
% Description:
% S is computed by using the formula:
% S = D2 + D2' - 2*X'*X
% where all entries of the i-th row of D2 have
% the same value of the (i,i)-entry of X'*X.
% Example:
% X = rand(200,400);
```

```
% S = square_distance(X);

% Check input.
if (nargin > 1)
    error('Too many input arguments.');
end
if ~isreal(X)
    error('Input matrix must be real.');
end
% Compute entry-wise square of D2.
D2 = repmat(sum(X.*X), size(X,2), 1);
% Compute matrix S.
S = D2 + D2' - 2*X'*X;
```

Sometimes, we need to compute the square Euclidean distances between two data sets, i.e., the distances of all pairs, which contain points from two different data sets $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ and $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_m\} \subset \mathbb{R}^D$, respectively. We represent all of the square distances by an $m \times n$ matrix $\mathbf{S} = [s_{ij}]$ such that

$$s_{ij} = d_2^2(\mathbf{y}_i, \mathbf{x}_j), \quad \mathbf{y}_i \in \mathcal{Y}, \quad \mathbf{x}_j \in \mathcal{X}.$$

The matrix \mathbf{S} can be computed by the similar method used in the previous algorithm. The Matlab code for the computation is the following.

```
function S = square_distance2m(Y,X)
% compute square Euclidean distance of 2 matrices.
% S = SQUARE_DISTANCE2M(Y,X)
% INPUTS:
% X: (D x n) matrix for the set of n points in R^D.
% Y: (D x m) matrix for the set of m points in R^D.
% OUTPUT:
% S: (m x n) matrix whose (i,j)-entry is the square
%     Euclidean distances from y_i to x_j.
% Description:
% S is computed by the formula:
% S = XD2 + YD2' - 2*Y'X
% where the entries of the j-th row of XD2 and
% the i-th row of YD2 are the square norms of x_j
% and y_i respectively.
% Example :
% X = rand(200,400); Y = (100,400)
% S = square_distance2m(Y, X);

% Check input.
if nargin > 2
    error('Too many input arguments.');
end
if ~isreal(Y) || isreal(X)
    error('Input matrices must be real.');
end
```

```

if size(Y,1)=size(X,1)
    error('Y and X must have the same # of rows.');
end
% Initialize inputs
if nargin==1
    X=Y;
end
% Compute square distances of X and Y.
Y2 = sum(Y.*Y); X2 = sum(X.*X);
% Compute matrix S.
S = repmat(X2, size(Y,2),1) + ...
    repmat(Y2',1,size(X,2)) - 2*Y'*X;

```

The next function is for the construction of the neighborhood system on the data. The output of the function consists of a sparse weight matrix **ND2** and the adjacency matrix **A** of the data graph (see the definition of data graph, adjacency matrix, and weight matrix in the next section). In the weight matrix **ND2**, the (i, j) -entry has the value $d_2^2(\mathbf{x}_i, \mathbf{x}_j)$ if \mathbf{x}_j is a neighbor of \mathbf{x}_i , and vanishes otherwise.

Remark 3.1. Mathematically, if \mathbf{x}_j is not a neighbor of \mathbf{x}_i , the distance from \mathbf{x}_j to \mathbf{x}_i should be set as infinity, which in Matlab has the syntax `Inf`. However, in order to make **ND2** sparse, we replace `Inf` by 0 in our code. When the matrix **ND2** is used by other algorithms, all non-diagonal 0 entries need to be converted to infinity.

```

function [ND2, A] = data_neighborhood(X,options)
% Construct neighborhood system on data.
% [ND2, A] = DATA_NEIGHBORHOOD(X, OPTIONS)
% INPUTS:
%   X: (D x n) matrix,
%       in which X(:,i) is the ith point in R^D.
% OPTIONS:
%   .epsilon(double) for epsilon-neighbors.
%   .k (integer or n-vector) for k-neighbors.
%   .symmetric (Boolean)
%       if =1, output a symmetric pre-weight matrix.
%       Otherwise, it may be asymmetric
%   .verb (Boolean) if =1, display the comments.
% OUTPUTS:
%   ND2 (double,sparse): sparse matrix, in which
%   each row contains only the square distances
%   from the i-th point to its neighbors. i.e.,
%   ND2(i,j)=0, if point j is not a neighbor of
%   the i-th point.
%   Otherwise, ND2(i,j) = S(i,j).
%   A (Boolean, sparse): the adjacency matrix.
%   A(i,j) = 1, if x_j is a neighbor of x_i.
%   Otherwise, A(i,j) = 0.

```

```
% Example.
% X = rand(200,400);
% options.k = 10;
% options.symmetric = 0;
% options.verb = 1;
% [ND2,A]= data_neighborhood(X, options);

% Initialize options.
options.null=0;
n = size(X,2);
emode = false;
if isfield(options, 'verb')
    verb = options.verb;
else
    verb = 1;
end
if isfield(options, 'epsilon')
    epsilon = options.epsilon;
    emode = true;
elseif isfield(options, 'k')
    k = options.k;
    % Check the validity of k
    if (numel(k) ~= 1)&&(numel(k) ~= n)
        error('k must be integer or n-vector');
    elseif numel(k)==1
        k=k*ones(1,n);
    else
        k=k(:);
    end
else
    k = 10*ones(1,n);
end
if isfield(options, 'symmetric')
    symmetric = options.symmetric;
else
    symmetric = 0;
end
% Compute square distance matrix.
if verb
    disp('- computing distance matrix.');
end
S = square_distance(X);
clear X
% Construct neighborhood and compute
% square distances for neighbors.
if emode % Compute eps-neighborhood.
    if verb
        disp('- computing eps-neighborhood.');
    end
    S(S> epsilon^2) = 0;
```

```

ND2 = sparse(S);
else % Computer k-neighborhood.
    if verb
        disp('- computing k-neighborhood.');
    end
    ND2 = sparse(n,n);
    [sortS, indexS] = sort(S);
    for i = 1:n
        nbri = indexS(2:k(i)+1,i);
        ND2(nbri,i) = sortS(2:k(i)+1,i);
    end
end
if symmetric
    ND2 = (ND2 + ND2')/2;
end
A = (ND2>0);

```

3.2 Graphs on Data Sets

The geometry of discrete data is described by the neighborhood system, which can be represented effectively by a graph. In mathematics, a graph is an abstract representation of a set of objects where some pairs of the objects are connected by links. A graph can be either *directed* or *undirected*. Their definitions are given below. Because undirected graph is more often used than directed graph, the term *graph* usually means *undirected graph*, while a directed graph is called a *digraph*. Although graphs can be constructed on either finite sets or infinite sets, in this book we only consider the graphs on finite sets.

3.2.1 Undirected Graphs

Definition 3.1. Let V be a given finite set. An undirected (or simple) graph G is an ordered pair $G = [V, E]$ so that the elements of E are 2-elements subsets (i.e., unordered pairs) of V . The elements of V are called *vertices* (or *nodes*, or *points*) of the graph G and the elements of E are called *edges* (or *lines*) of G . If $E = \emptyset$, then G is called an edgeless (or trivial) graph.

Many real-world problems can be modeled by graphs. Figure 3.2 illustrates a graph model for the famous problem of the Seven Bridges of Königsberg.

Later, we shall write the vertex set and the edge set as $V(G)$ and $E(G)$ respectively if the relation between V (or E) and G has to be stressed. Let $a, b \in V$ be two vertices. We denote the edge connecting them by (a, b) and call a and b the ends (or endpoints, or end vertices) of the edge. We also say

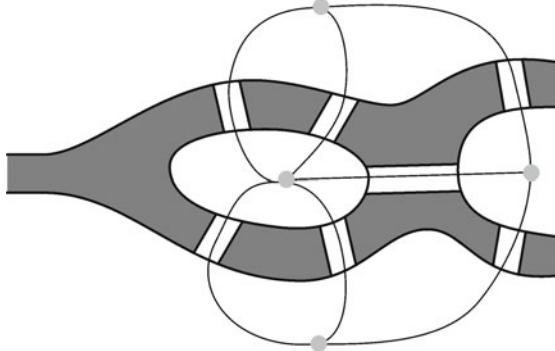


Fig. 3.2 A graph model of the Seven Bridges of Königsberg. The four places separated by rivers are represented by nodes, and each bridge is represented by an edge. The problem of the Seven Bridges of Königsberg is modeled by an undirected graph.

that a and b are adjacent. In an undirected graph, (a, b) and (b, a) represent the same edge. Sometimes we need the edge set containing the self edges (or loops). A loop, denoted by (a, a) , is the edge that starts and ends on the same vertex a . Since each loop uniquely corresponds to a vertex, the set of loops will also be denoted by V . Therefore, the graph with loops is denoted by $\bar{G} = [V, E \cup V]$. When stated without any qualification, a graph is assumed to be simple (i.e., without loops). In this book, we do not use the graphs with loops.

In a simple graph G , the number of edges that connect the vertex a is called the degree of a and denoted by d_a . Maximum and minimum degrees over G are denoted by $\Delta(G)$ and $\delta(G)$ respectively. The set of all vertices that are connected to a by edges is called the neighborhood of a and denoted by $N_G(a)$. We agree that $a \notin N_G(a)$. The extended neighborhood is defined by $\bar{N}_G(a) = N_G(a) \cup \{a\}$ that contains a itself. In all of these notations, when the context is clear, the subscript G will be deleted. A graph is called regular if every vertex has the same degree, i.e., $\Delta = \delta$. A regular graph with vertices of degree k is called a k -regular graph or regular graph of degree k . A graph is called complete if each pair of nodes has an edge connecting them. Hence, a complete graph is a regular graph of degree $|V| - 1$, where $|V|$ is the cardinal number of the vertex set V .

A subgraph $G_1 = [V_1, E_1]$ of a (undirected) graph $G = [V, E]$ is a (undirected) graph such that $V_1 \subset V$, and E_1 only contains the edges (in E) whose ends are in V_1 . A linear graph (of length n), denoted by $P = [V_P, E_P]$, is the graph such that the vertices in V_P can be listed in an order, say, v_0, v_1, \dots, v_n , and

$$E_P = \{(v_{i-1}, v_i)\}, \quad i = 1, 2, \dots, n.$$

If a linear graph P is a subgraph of another graph G , P is called a path from $a = v_0$ to $b = v_n$ in G and denoted by $\gamma(a, b)$. It is obvious that in an undirected graph, a path from a to b is also a path from b to a : $\gamma(a, b) = \gamma(b, a)$. Two vertices a and b are called connected if G contains a path from a to b . Otherwise, they are called disconnected. A graph is called connected if every pair of distinct vertices in the graph is connected. Otherwise, it is called disconnected. A graph is called k -vertex-connected (k -edge-connected) if and only if the removal of a set with k vertices (k edges) makes the graph disconnected. A k -vertex-connected graph is often simply called k -connected.

If a graph $G = [V, E]$ is disconnected, then there exist m mutually disjoint subsets of V , say V_1, \dots, V_m , and m mutually disjoint subsets of E , say E_1, \dots, E_m , such that $\cup_{i=1}^m V_i = V$, $\cup_{i=1}^m E_i = E$, and each $G_i = [V_i, E_i], 1 \leq i \leq m$, is a connected graph. In this case, we say that $G_i, 1 \leq i \leq m$, are m connected components of G .

Two graphs G and H are said to be isomorphic, denoted by $G \sim H$, if there is a one-to-one correspondence, called isomorphism, from $V(G)$ to $V(H)$ such that two vertices are adjacent in G if and only if their corresponding vertices are adjacent in H . Two isomorphic graphs are also called equivalent graphs. Likewise, a graph G is said to be homomorphic to a graph H if there is a mapping, called a homomorphism, from $V(G)$ to $V(H)$ such that if two vertices are adjacent in G then their corresponding vertices are adjacent in H .

Let $G = [V, E]$ be a graph with $n = |V|$. We may construct a graph $H = [V(H), E(H)]$ such that H is equivalent to G and $V(H) = \{1, 2, \dots, n\}$. Then the isomorphism T from G to H is called a graph labeling for G , and the graph H is called the label graph of G . For the simplification of notations, we shall identify $G = [V, E]$ with an equivalent label graph, and agree that an edge in E , say $(v_i, v_j) \in E$, is denoted by a pair (i, j) .

We often use adjacency matrix to represent the edge set in a graph. The adjacency matrix \mathbf{A} of a graph $G = [V, E]$ is the $n \times n$ (symmetric) matrix defined by

$$\mathbf{A}(i, j) = \begin{cases} 1, & \text{if } (i, j) \in E \text{ or } (j, i) \in E, \\ 0, & \text{otherwise.} \end{cases} \quad (3.5)$$

The edge set of a graph is uniquely determined by its adjacency matrix. In mathematical computation, using the adjacency matrix is more convenient than using edge set. Hence, we often rewrite $G = [V, E]$ by $G = [V, \mathbf{A}]$. Note that the diagonal entries of the adjacency matrix are 0 for a simple graph, but 1 for a graph with loops. The weight matrix is a generalization of the adjacency matrix. In a weight matrix the (i, j) -entry, called a weight, usually is a positive real number. Thus, a weight matrix $\mathbf{W} = [w_{i,j}]_{n \times n}$ of a graph $G = [V, E]$ is a symmetric matrix with $w_{i,j} \neq 0$, if $(i, j) \in E$, and $w_{i,j} = 0$, otherwise. A graph equipped with a weight matrix is called a weighted graph. Although the adjacent matrix can be considered as a weight matrix with the

weight 1 for all edges, traditionally, we call G a weighted graph if $\mathbf{W} \neq \mathbf{A}$. We denote the weighted graph by $G = [V, \mathbf{W}]$ and denote the unweighted graph by $G = [V, E]$ or $G = [V, \mathbf{A}]$. The adjacency matrix of a weighted graph can be simply derived from the weight matrix $\mathbf{W}(G)$.

Assume that a disconnected graph G has m connected components G_1, \dots, G_m with the adjacency matrices $\mathbf{A}_1, \dots, \mathbf{A}_m$, respectively. Then the adjacency matrix of G can be written as the block-diagonal matrix

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \cdots & 0 \\ 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & \mathbf{A}_m \end{bmatrix}, \quad (3.6)$$

provided the vertices are properly ordered. In general, we need to permute \mathbf{A} by a permutation \mathbf{P} , making the matrix $\mathbf{P}'\mathbf{A}\mathbf{P}$ block-diagonal as in the form of (3.6).

In the context, we always identify the data graph $G = [\mathcal{X}, \mathbf{A}]$ with the labeling graph $[V, \mathbf{A}]$ by mapping x_i to i . It is clear that the neighborhood of a node $i \in V$ is the node set

$$N(i) = \{j \in V, \mathbf{A}(i, j) = 1\},$$

and the degree of the node i is $|N(i)|$. If $G = [V, \mathbf{W}]$ is a weighted graph (either simple or with loops), then the node volume of i is defined as the positive number

$$d_i = \sum_{j \in \bar{N}_i} w_{ij},$$

where $\bar{N}_i = N_i \cup \{i\}$. For a simple graph G , $w(i, i) = 0$. A subset $C \subset V$ is called a cluster of G . The volume of a cluster is defined by

$$\text{vol}(C) = \sum_{i \in C} d_i$$

and its boundary volume is defined by

$$\text{vol}(\partial C) = \sum_{i \in C, j \notin C} w_{ij}.$$

A cluster C generates a subgraph $G_C = [C, E_C]$ of G , where the edge set is derived by

$$E_C = \{(i, j) \in E : i, j \in C\}.$$

It is clear that the adjacency matrix of G_C , denoted by \mathbf{A}_C , is a submatrix of \mathbf{A} extracted from C rows and C columns of \mathbf{A} .

3.2.2 Directed Graphs

In applications, sometimes we also need digraphs.

Definition 3.2. A set pair $G = [V, E]$ is called a digraph (or directed graph) if V is a node set and E is a set of ordered pairs of two nodes in V , called *arcs*.

An arc in a digraph is also called a directed edge or an arrow. Note that for two nodes $a, b \in V$, (a, b) and (b, a) define two different arcs of G . The arc (b, a) is called the symmetric arc (or the inverted arc) of (a, b) . An oriented graph is a digraph having no symmetric arcs. In a digraph $G = [V, E]$, an arc $e = (a, b) \in E$ is considered to be directed from a to b , where b is called the head and a called the tail. We also call b a direct successor of a and a a direct predecessor of b . A loop (a, a) in a digraph keeps a sense of direction, treating both head and tail identically. When stated without any qualification, a digraph is assumed to have no loops.

The isomorphism and labeling of digraphs can be defined in the similar way as for undirected graphs. We omit the details of their definitions. Later, we always assume that a digraph is well labeled so that the node set is identified with its labeling set.

We can employ the adjacency matrix to describe a digraph G too.

Definition 3.3. Let $G = [V, E]$ be a (label) digraph with $V = \{1, 2, \dots, n\}$. The adjacency matrix of G is the $n \times n$ matrix such that

$$\mathbf{A}(i, j) = \begin{cases} 1, & \text{if } (i, j) \in E, \\ 0, & \text{otherwise.} \end{cases} \quad (3.7)$$

It is clear that the adjacency matrix \mathbf{A} uniquely determines the (label) digraph G . The notation $G = [V, \mathbf{A}]$ can denote either a graph or a digraph, noting that the adjacency matrix of a digraph may be asymmetric. If G is simple, then all $\mathbf{A}(i, i) = 0$; else if G contains a loop (i, i) , then $\mathbf{A}(i, i) = 1$. For an oriented graph at least one of $\mathbf{A}(i, j)$ and $\mathbf{A}(j, i)$, $(i \neq j)$ vanishes. Assume that G is a simple digraph. Let \mathbf{A}^u be the upper-triangular matrix and \mathbf{A}^l be the lower-triangular matrix such that $\mathbf{A}^u + \mathbf{A}^l = \mathbf{A}$. Let E^u and E^l be two arc sets derived from \mathbf{A}^u and \mathbf{A}^l respectively. Then both $G^u = [V, E^u]$ and $G^l = [V, E^l]$ are oriented graphs. At the vertex i of a digraph G , the *out-degree* $d_G^+(i)$ is the number of arcs starting from i , and the *in-degree* $d_G^-(i)$ is the number of arcs ending at i . The degree of i is then defined as $d_G(i) = d_G^+(i) + d_G^-(i)$. We can formulate $d_G^+(i)$ and $d_G^-(i)$ as

$$\begin{aligned} d_G^+(i) &= |N_G^+(i)|, \\ d_G^-(i) &= |N_G^-(i)|, \end{aligned}$$

where $N_G^+(i)$ is the set of tails of arcs going apart from i , called the out-neighborhood (or successor set) of the vertex i , and, likewise, $N_G^-(i)$ is the

set of heads of arcs going into i , called the in-neighborhood (or predecessor set) of the vertex i . It is obvious that

$$\begin{aligned} N_G^+(i) &= \{j: \mathbf{A}(i, j) = 1\}, \\ N_G^-(i) &= \{j: \mathbf{A}(j, i) = 1\}. \end{aligned}$$

We shall omit the subscript G in the notations above when the context is clear. Let maximum and minimum out-degrees be denoted by Δ^+ and δ^+ respectively, and maximum and minimum in-degrees by Δ^- and δ^- respectively. Their definitions are similar to the maximum and minimum degrees of a graph. We can also define the regular digraph, regular digraph of out-degree k , regular digraph of in-degree k , and so on. Since these definitions are similar to those for undirected graph, we omit the details here.

A subgraph of a digraph G is a digraph whose vertex set is a subset of that of G , and whose adjacency relation is a subset of E restricted to this vertex subset. A linear digraph (of length n) $P = [V_P, E_P]$ is the digraph such that the vertices in V_P can be listed in an order, say, v_0, v_1, \dots, v_n such that

$$E_P = \{(v_{i-1}, v_i)\}, \quad i = 1, 2, \dots, n.$$

If a linear digraph P occurs as a subgraph of another digraph G , P is called a directed path from $a = v_0$ to $b = v_n$ in G and denoted by $\gamma(a, b)$. In a digraph, the existence of a path from a to b does not imply the existence of a path from b to a . A directed path from b to a is called an inverted path of $\gamma(a, b)$ if it has the same vertex set and the inverted arcs of $\gamma(a, b)$. Note that in a digraph, a directed path may have no inverse.

A digraph $G = [V, \mathbf{A}]$ is called symmetric if \mathbf{A} is symmetric. It is clear that a symmetric digraph $G = [V, \mathbf{A}]$ is equivalent to an undirected graph $G = [V, \mathbf{A}]$. In this sense, an undirected graph is a special digraph.

A digraph is called weakly connected, when all of its directed edges are replaced by undirected edges, a connected graph will be produced. It is called strongly connected if for every pair of vertices u and v in V , the digraph containing a directed path from u to v also contains a directed path from v to u .

We now define the weight matrix for a digraph.

Definition 3.4. Let $G = [V, \mathbf{A}]$ be a digraph with $V = \{1, 2, \dots, n\}$. A weight matrix $\mathbf{W} \stackrel{\text{def}}{=} [\mathbf{W}_{i,j}]$ on G is defined by

$$\mathbf{W}_{i,j} = \begin{cases} w_{ij}, & \text{if } \mathbf{A}(i, j) = 1, \\ 0, & \text{otherwise,} \end{cases} \quad (3.8)$$

where $w_{ij} > 0$ is the weight on the arc (i, j) .

Note that \mathbf{W} may be asymmetric even though \mathbf{A} is symmetric. The adjacency (or weight) matrix provides a uniform representation for both digraph

and undirected graph: An unweighted graph is undirected if and only if its adjacency matrix is symmetric, and a weighted graph is undirected if and only if its weight matrix is symmetric. On these grounds, we shall use the term “graph” for both undirected graph and digraph, if the adjacency matrix or the weight matrix associated with a graph is presented, although we agreed early that the term “graph” stands for the undirected graph only.

We now define volumes of a node or a cluster in digraph.

Definition 3.5. Let $G = [V, \mathbf{W}]$ be a weighted digraph without loops. Let the out-neighborhood and the in-neighborhood of a node $i \in V$ be denoted by $N^+(i)$ and $N^-(i)$ respectively. Then, the out-volume and in-volume of i are defined by $d_i^+ = \sum_{j \in N^+(i)} w_{i,j}$ and $d_i^- = \sum_{j \in N^-(i)} w_{j,i}$, respectively. Similarly, the out-volume and in-volume of a cluster $C \subset V$ are defined by $\text{vol}^+(C) = \sum_{i \in C} d_i^+$ and $\text{vol}^-(C) = \sum_{i \in C} d_i^-$, respectively, and its boundary out-volume and boundary in-volume are defined as $\text{vol}^+(\partial C) = \sum_{i \in C, j \notin C} \mathbf{W}_{i,j}^+$ and $\text{vol}^-(\partial C) = \sum_{i \in C, j \notin C} \mathbf{W}_{j,i}^-$, respectively.

3.2.3 Neighborhood and Data Graphs

The data graph is an effective tool in processing data geometry. Let $\mathcal{X} \subset \mathbb{R}^D$ be a data set. Two main methods for defining neighborhood systems on data are the k -neighborhood and the ε -neighborhood. A neighborhood system on a data set uniquely determines the data graph $G = [\mathcal{X}, \mathbf{A}]$ so that $\mathbf{A}(i, j) = 1$ if and only if \mathbf{x}_j is a neighbor of \mathbf{x}_i . A k -neighborhood system on \mathcal{X} determines a regular digraph (of out-degree k). Since the relation between the neighborhood system and the adjacency matrix of a data graph is unique, each (directed or undirected) graph $G = [\mathcal{X}, \mathbf{A}]$ uniquely determines a neighborhood system on \mathcal{X} . Sometimes, we also need a k -neighborhood system on \mathcal{X} with $k = k(i)$ varying on i . In some dimensionality reduction methods, a symmetric adjacency matrix is required. In this case, if a k -neighborhood system is defined on \mathcal{X} , we need to symmetrize the adjacency matrix \mathbf{A} . For this purpose, we may re-set the adjacency matrix as $\tilde{\mathbf{A}} = \max(\mathbf{A}, \mathbf{A}')$ or $\tilde{\mathbf{A}} = \min(\mathbf{A}, \mathbf{A}')$, where the maximum or minimum is taken entrywise. An ε -neighborhood system on \mathcal{X} always determines an undirected graph. When the ε -neighborhood method is applied, in order to derive a nontrivial graph, the restriction $\varepsilon > d$ is required, where d is the minimal distance over \mathcal{X} :

$$d = \min\{d_2(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in \mathcal{X}, \mathbf{x} \neq \mathbf{y}\}.$$

Since a data set equipped with a neighborhood system is equivalent to a graph on the data, the Matlab function `data_neighborhood.m` that creates neighborhood system on data (see Subsection 3.1.2) also generates a data graph.

3.3 Spectral Analysis of Graphs

In the last section, we used data graphs to study the geometry of data sets. The subject of *spectral graph theory* is to find modes of data with the aid of the spectral analysis of certain operators on graphs. In spectral graph analysis, the adjacency matrix is one of the most important operators on graphs. Another important operator that closely relates to the adjacency operator is the Laplacian.

3.3.1 Laplacian of Graphs

Definition 3.6. Let $G = [V, \mathbf{A}]$ be a graph, where V is a node set with $|V| = n$ and \mathbf{A} is the adjacency matrix. The (non-normalized) Laplacian matrix \mathbf{L}_G on the graph G is defined by

$$\mathbf{L}_G(i, j) = \begin{cases} -1, & \mathbf{A}(i, j) = 1, \\ d_i, & i = j, \\ 0, & \text{otherwise,} \end{cases} \quad (3.9)$$

where d_i is the degree of the i th node.

Let \mathbf{D} denote the diagonal matrix with $\mathbf{D}_{i,i} = d_i$. The Laplacian matrix in (3.9) can be written as

$$\mathbf{L} (= \mathbf{L}_G) = \mathbf{D} - \mathbf{A}.$$

The operator defined in (3.9) usually is normalized to the following.

$$\mathcal{L}(i, j) = \begin{cases} -\frac{1}{\sqrt{d_i d_j}}, & \mathbf{A}(i, j) = 1, \\ 1, & i = j \text{ and } d_i \neq 0, \\ 0, & \text{otherwise.} \end{cases}$$

We have

$$\mathcal{L} = \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}, \quad (3.10)$$

with the convention $\mathbf{D}^{-1}(i, i) = 0$ if $d_i = 0$. Recall that $d_i = 0$ if and only if the i th node is an isolated vertex, to which no vertices connect. The operator \mathcal{L} is called the Laplacian of G . When G is k -regular, $\mathbf{D}^{-1}(i, i) = 1/k$, and therefore,

$$\mathcal{L} = \mathbf{I} - \frac{1}{k} \mathbf{A},$$

where \mathbf{I} is the identity matrix. The Laplacian \mathcal{L} can be considered as a discrete version of the Laplacian $L = -\Delta$ on the space of twice differential

functions. Indeed, let $H(V)$ be the space of (discrete) functions $g : V \rightarrow \mathbb{R}$. Then \mathcal{L} is the operator on $H(V)$ such that

$$\mathcal{L}g(i) = \frac{1}{\sqrt{d_i}} \sum_{j \in N(i)} \left(\frac{g(i)}{\sqrt{d_i}} - \frac{g(j)}{\sqrt{d_j}} \right),$$

where the weighted difference on the right-hand side is the discrete form of the directional derivative of g .

We confirm that (the discrete operator) \mathcal{L} is self-joint.

Theorem 3.1. *The Laplacian \mathcal{L} of $G = [V, E]$ is self-joint. That is, there is a matrix \mathbf{S} such that*

$$\mathcal{L} = \mathbf{S}' \mathbf{S}. \quad (3.11)$$

Proof. Assume $|V| = n$ and V is represented by the label set $\{1, 2, \dots, n\}$. Let C be the collection of all 2-element sets in V . Then the cardinal number of C is $s = n(n-1)/2$. Let each element in C be denoted by (i, j) with $i < j$. Then $E \subset C$. There is a one-to-one and onto mapping $L : C \rightarrow \{1, 2, \dots, s\}$, labeling the elements of C . For example,

$$L((i, j)) = (i-1) \left(n - \frac{i}{2} \right) + j - i$$

defines such a mapping, whose inverse L^{-1} maps an integer $k (1 \leq k \leq s)$ to a pair (i, j) , $1 \leq i < j \leq n$. We now construct the $s \times n$ matrix \mathbf{S} by

$$\mathbf{S}(k, i) = \begin{cases} \frac{1}{\sqrt{d_i}}, & \text{if } L^{-1}(k) = (i, j) \in E \text{ for some } j \in V, j > i, \\ -\frac{1}{\sqrt{d_j}}, & \text{if } L^{-1}(k) = (j, i) \in E \text{ for some } j \in V, j < i, \\ 0, & \text{otherwise.} \end{cases}$$

It can be verified that $\mathcal{L} = \mathbf{S}' \mathbf{S}$. The theorem is proved. \square

Since \mathcal{L} has the decomposition (3.11), its eigenvalues are real and non-negative. The eigenvalue set $\{\lambda_i\}_{i=0}^{n-1}$ of \mathcal{L} is called the spectrum of \mathcal{L} (sometimes also called the spectrum of the graph G). We are interested in the basic properties of the spectrum. It is easy to see that 0 is an eigenvalue of \mathcal{L} and $v_0 = \mathbf{D}^{1/2} \mathbf{1}$ is an 0-eigenvectors of \mathcal{L} , where $\mathbf{1}$ denotes the n -vector whose entries are 1. Let the eigenvalues of \mathcal{L} be ordered as $0 = \lambda_0 \leq \lambda_1 \leq \dots \leq \lambda_{n-1}$. To estimate their bounds, we establish the following formulas.

Let $g \in H(V)$, $g \neq 0$, and $f = D^{-1/2}g$, where D is the diagonal mapping such that $Dg_i = d_i g_i$. Then

$$\begin{aligned} \frac{\langle g, \mathcal{L}g \rangle}{\langle g, g \rangle} &= \frac{\langle g, D^{-1/2}LD^{-1/2}g \rangle}{\langle g, g \rangle} = \frac{\langle f, Lf \rangle}{\langle D^{1/2}f, D^{1/2}f \rangle} \\ &= \frac{\sum_{(i,j) \in E} (f(i) - f(j))^2}{\sum_{i=1}^n f^2(i)d_i}. \end{aligned} \quad (3.12)$$

The sum $\sum_{(i,j) \in E} (f(i) - f(j))^2$ is called the Dirichlet sum of G and the ratio in (3.12) the Rayleigh quotient. By (3.12), we have

$$\lambda_G (\stackrel{\text{def}}{=} \lambda_1) = \inf_{g \perp v_0} \frac{\langle g, \mathcal{L}g \rangle}{\langle g, g \rangle} = \inf_{f \perp D^{1/2}v_0} \frac{\sum_{(i,j) \in E} (f(i) - f(j))^2}{\sum_{i=1}^n f^2(i)d_i}. \quad (3.13)$$

The function f solving (3.13) is called the harmonic eigenfunction of G . It is easy to see that if g is an eigenfunction of \mathcal{L} achieving the eigenvalue λ , then $f = D^{-1/2}g$ is an eigenfunction of the following generalized eigen-problem:

$$Lf = \lambda Df, \quad (3.14)$$

achieving the same λ .

We return to the discussion on the eigenvalues of \mathcal{L} . Denote by V_k the eigenspace spanned by the eigenvectors achieving $\lambda_0, \lambda_1, \dots, \lambda_k$, ($1 \leq k \leq n-1$), and denote by V_k^\perp the o.g. complement of V_k with respect to \mathbb{R}^n . Then, V_k^\perp is the eigenspace spanned by the eigenvectors achieving $\lambda_{k+1}, \dots, \lambda_{n-1}$. By Courant-Fischer Theorem, we have

$$\lambda_k = \inf_{f \perp D^{1/2}(V_{k-1})} \frac{\sum_{(i,j) \in E} (f(i) - f(j))^2}{\sum_{i=1}^n f^2(i)d_i} \quad (3.15)$$

and, alternately,

$$\lambda_k = \sup_{f \perp D^{1/2}(V_k^\perp)} \frac{\sum_{(i,j) \in E} (f(i) - f(j))^2}{\sum_{i=1}^n f^2(i)d_i}. \quad (3.16)$$

The function f achieving the infimum (supremum) above is called a harmonic eigenfunction of G achieving λ_k . Recall that f is a harmonic eigenfunction of

G (achieving λ_k) if and only if $g = D^{1/2}f$ is an eigenfunction of \mathcal{L} (achieving λ_k).

The above formulation for eigenvalues of \mathcal{L} corresponds in a natural way to the eigenvalues of the Laplace-Beltrami operator \mathcal{L}_M on a Riemannian manifold M with the homogeneous Neumann boundary condition. Let e denote the constant function having the value 1 over M . It is clear that the function e satisfies the homogeneous Neumann boundary condition and $\mathcal{L}_M e = 0$. Hence, it is a 0-eigenvector of \mathcal{L}_M . A function $f \in H(M)$ satisfying $\int_M f = 0$ can be written as $\langle f, e \rangle = 0$ (or $f \perp e = 0$). Then the eigenvalue λ_1 of \mathcal{L}_M can be computed by

$$\lambda_1 = \inf_{f \perp e} \frac{\int_M \langle \text{grad } f, \text{grad } f \rangle}{\int_M |f|^2}.$$

The formula for the computation of a general eigenvalue λ_k of \mathcal{L}_M can be developed in the similar way as in (3.15) and (3.16).

The spectrum of \mathcal{L} on a graph describes some geometric properties of the graph. The following lemma gives a preliminary result in this aspect.

Lemma 3.1. *Let \mathcal{L} be the Laplacian of a graph $G = [V, E]$ with $|V| = n$. Let $\{\lambda_i\}_{i=0}^{n-1}$ be the spectrum of \mathcal{L} . We have the following:*

- (1) *The sum of all eigenvalues of \mathcal{L} satisfies*

$$\sum_{i=0}^{n-1} \lambda_i \leq n, \quad (3.17)$$

where the equality holds if and only if G has no isolated vertices. Furthermore, if G has no isolated vertices, then

$$\lambda_{n-1} \geq \frac{n}{n-1}.$$

- (2) *For any graph G with $n \geq 2$,*

$$\lambda_G \leq \frac{n}{n-1}, \quad (3.18)$$

where the equality holds if and only if G is a complete graph. Furthermore, $\lambda_G \leq 1$ if and only if G is not complete, and $\lambda_G > 0$ if and only if G is connected.

- (3) *For all i , $1 \leq i \leq n-1$,*

$$\lambda_i \leq 2.$$

- (4) *The graph G has exactly $k+1$ connected components if and only if $\lambda_0 = \dots = \lambda_k = 0$, but $\lambda_{k+1} > 0$. In this case, the spectrum of G is the union of the spectra of its connected components.*

Proof. (1) We have $\sum_{i=0}^{n-1} \lambda_i = \text{tr}(\mathcal{L}) \leq n$. Recall that $\mathcal{L}(i, i) = 1$ or 0 and it vanishes if and only if the i th node is an isolated vertex. Hence, $\sum_{i=0}^{n-1} \lambda_i = n$ if and only if G has no isolated vertices. Besides, since $\lambda_0 = 0$ and λ_{n-1} is the largest eigenvalue, we have $\lambda_{n-1} \geq \frac{n}{n-1}$.

(2) The inequality (3.18) directly follows (3.17) and $\lambda_0 = 0$. We now assume G is complete. Then

$$\mathcal{L} = \frac{n}{n-1} \mathbf{I} - \frac{1}{n-1} \mathbf{1} \mathbf{1}^T,$$

where the matrix $\mathbf{1} \mathbf{1}^T$ has only one non-vanished eigenvalue n . It follows that $\lambda_0 = 0$ and $\lambda_i = \frac{n}{n-1}$ for $i \geq 1$. On the other hand, if $\lambda_0 = 0$ and $\lambda_i = \frac{n}{n-1}$ for $i \geq 1$, then, by $\mathcal{L} = \mathbf{I} - \frac{1}{n-1} \mathbf{A}$, the adjacency matrix \mathbf{A} must be in the form of $\mathbf{1} \mathbf{1}^T - \mathbf{I}$, i.e., G is complete. We now discuss the case that G is incomplete. If G is disconnected too, then $\lambda_1 = 0$ (see the proof for (4)) so that $\lambda_1 \leq 1$ trivially holds. Assume that G is incomplete but connected. Then there are $i, j \in V, i \neq j$, so that $(i, j) \notin E$, i.e., $\mathbf{A}(i, j) = \mathbf{A}(j, i) = 0$. We define the function f by

$$f(k) = \begin{cases} d_j, & \text{if } k = i, \\ -d_i, & \text{if } k = j, \\ 0, & \text{otherwise.} \end{cases}$$

Since $D^{1/2}v_0 = [d_1, \dots, d_n]'$, we have $\langle f, D^{1/2}v_0 \rangle = 0$. We also have, by $\mathbf{A}(i, j) = \mathbf{A}(j, i) = 0$,

$$\sum_{(i,j) \in E} (f(i) - f(j))^2 = d_i f^2(i) + d_j f^2(j) = \sum_{i=1}^n f^2(i) d_i.$$

By (3.13), $\lambda_1 \leq 1$. We now prove the statement that $\lambda_1 > 0$ if and only if G is connected. Let f be a harmonic eigenfunction of G achieving 0. Then

$$0 = \langle g, \mathcal{L}g \rangle = \sum_{(i,j) \in E} (f(i) - f(j))^2. \quad (3.19)$$

The constant function $f = \text{const}$ is a harmonic 0-eigenfunction of (3.19). If (3.19) has the only solution $f = \text{const}$ (i.e. 0 is a simple eigenvalue), then G must be connected. Otherwise, without loss of generality, we assume that G has a connected component $G_1 = [V_1, E_1]$ such that $|V_1| > 1$ and $V_1 \neq V$. We define

$$h(k) = \begin{cases} 1, & \text{if } k \in V_1, \\ 0, & \text{if } k \notin V_1. \end{cases} \quad (3.20)$$

Then h is also a harmonic 0-eigenvector of \mathcal{L} and $\{f, h\}$ is a linearly independent set. Hence, 0 is at least a double eigenvalue of \mathcal{L} , i.e., $\lambda_G = 0$. On the other hand, if G has a multiple 0 eigenvalue, by the similar argument

above, the equation (3.19) must have a solution h with the form of (3.20), where V_1 is a proper subset of V and disconnected to $V \setminus V_1$. Therefore, V is disconnected.

(3) For each $i, 1 \leq i \leq n - 1$, we may choose a function $f \neq 0$ and $f \perp D^{1/2}(V_{i-1})$. By (3.12) and the inequality

$$(f(i) - f(j))^2 \leq 2(f^2(i) + f^2(j)),$$

we have

$$\lambda_i \leq \frac{\sum_{(i,j) \in E} (f(i) - f(j))^2}{\sum_{i=1}^n f^2(i) d_i} \leq 2.$$

(4) The proof follows the definition of connected graphs, (3.6), and the fact that a graph is connected if and only if 0 is a simple eigenvalue of \mathcal{L} . \square

In the spectral graph theory, λ_G is perhaps the most important eigenvalue of a connected graph. In Lemma 3.1, we already know that for a connected incomplete graph, $0 < \lambda_G \leq 1$. We now try to obtain better estimates for its lower bound and upper bound. The estimate of λ_G heavily relies on the geometric properties of the graph. We first give an estimate of its low bound in terms of the volume and diameter of the graph. For two vertices i and j in an unweighted graph G , the graph distance between i and j is defined by the number of edges in the shortest path joining i and j . The diameter of G is defined by the maximum graph distance on G .

Lemma 3.2. *For a connected graph G with diameter D , we have*

$$\lambda_G \geq \frac{1}{D \operatorname{vol}(G)}.$$

Proof. Assume that f is a normalized harmonic eigenfunction of G . Let i_0 denote the vertex with

$$|f(i_0)| = \max_{i \in V} |f(i)|.$$

Since the constant e is the harmonic eigenfunction achieving 0-eigenvalue, $f \perp De$, i.e., $\sum_i d_i f(i) = 0$. Therefore, there exists a vertex j_0 satisfying $f(i_0)f(j_0) < 0$. Let P denote a shortest path in G connecting i_0 and j_0 .

By (3.12) and the Cauchy-Schwartz inequality, we have

$$\begin{aligned}\lambda_G &= \frac{\sum_{(i,j) \in E} (f(i) - f(j))^2}{\sum_{i=1}^n f^2(i) d_i} \\ &\geq \frac{\sum_{(i,j) \in P} (f(i) - f(j))^2}{\text{vol}(G) f^2(i_0)} \\ &\geq \frac{(f(i_0) - f(j_0))^2}{D \text{vol}(G) f^2(i_0)} \\ &\geq \frac{1}{D \text{vol}(G)}.\end{aligned}$$

The lemma is proved. \square

The following lemma gives a local characterization of the spectrum of \mathcal{L} .

Lemma 3.3. *Let f^k denote the harmonic eigenfunction achieving λ_k . Then, for each vertex $i \in V$, we have*

$$\sum_{j \in N(i)} (f^k(i) - f^k(j)) = \lambda_k d_i f^k(i). \quad (3.21)$$

Proof. Let \mathbf{A} be the adjacency matrix of the graph G and L be the Laplacian matrix. We have

$$Lf^k = \lambda_k Df^k, \quad 0 \leq k \leq n-1.$$

Extracting the i th components from the vectors on both sides of the equation above, we have

$$d_i f^k(i) - \sum_{j \in N(i)} f^k(j) = \lambda_k d_i f^k.$$

Since $d_i = |N(i)|$, we have

$$\lambda_k d_i f^k = d_i f^k(i) - \sum_{j \in N(i)} f^k(j) = \sum_{j \in N(i)} (f^k(i) - f^k(j)).$$

The lemma is proved. \square

Assume that G has no isolated vertex. Then $d_i > 0$ for each $i \in V$ and (3.21) can be written as

$$\lambda_k = \frac{1}{d_i f^k(i)} \sum_{j \in N(i)} (f^k(i) - f^k(j)).$$

3.3.2 Laplacian on Weighted Graphs

The definition of Laplacian matrix of a weighted graph is quite similar to that for an unweighted graph.

Definition 3.7. Let $G = [V, \mathbf{W}]$ be a weighted graph possibly with self-edges. The weighted Laplacian matrix \mathbf{L} on G is defined by

$$\mathbf{L}(i, j) = \begin{cases} -w_{i,j}, & (i, j) \in E, i \neq j \\ d_i - w_{i,i}, & i = j, \\ 0, & \text{otherwise,} \end{cases} \quad (3.22)$$

where d_i is the volume of i .

Let f be a function from V to \mathbb{R} . We have

$$\mathbf{L}f(i) = \sum_{k \in N(i)} (f(i) - f(k))w_{ik}.$$

Let \mathbf{D} denote the diagonal matrix $\mathbf{D} = \text{diag}(d_1, \dots, d_n)$. The Laplacian of the graph G is defined by the similar formula as (3.10):

$$\mathcal{L}(i, j) = \begin{cases} 1 - \frac{w_{i,i}}{d_i}, & \text{if } i = j \text{ and } d_i \neq 0, \\ -\frac{w_{i,j}}{\sqrt{d_i d_j}}, & \text{if } i \text{ and } j \text{ are adjacent,} \\ 0, & \text{otherwise,} \end{cases}$$

which leads to

$$\mathcal{L} = \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}.$$

By the similar derivation in (3.13), we have

$$\lambda_G = \inf_{\sum f(i)d_i=0} \frac{\sum_{(i,j) \in E} (f(i) - f(j))^2 w_{ij}}{\sum_{i=1}^n f^2(i)d_i}. \quad (3.23)$$

In general, denoting by V_k the eigenspace spanned by the eigenfunctions that achieve $\lambda_0, \lambda_1, \dots, \lambda_k$, ($1 \leq k \leq n-1$), respectively, we have

$$\lambda_k = \inf_{f \perp \mathbf{D}^{1/2}(V_{k-1})} \frac{\sum_{(i,j) \in E} (f(i) - f(j))^2 w_{ij}}{\sum_{i=1}^n f^2(i)d_i}$$

and, alternately,

$$\lambda_k = \sup_{f \perp \mathbf{D}^{1/2}(V_k^\perp)} \frac{\sum_{(i,j) \in E} (f(i) - f(j))^2 w_{ij}}{\sum_{i=1}^n f^2(i) d_i}.$$

3.3.3 Contracting Operator on Weighted Graph

We may apply contracting operator on a weighted graph $G = [V, \mathbf{W}]$ to building a new graph. Let $u, v \in V$ be two vertices. We denote the weight on the edge $(u, v) \in E$ by $w(u, v)$. A contracting operator $C_{u,v}$ merges u and v into a single vertex $u*$ such that for each $x \in V$, the new weights are computed by

$$\begin{aligned} w(x, u*) &= w(x, u) + w(x, v), \\ w(u*, u*) &= w(u, u) + w(v, v) + 2w(u, v). \end{aligned}$$

A graph built by several successive steps of contracting from G is called a contraction of G . The following theorem gives an estimate for the graph eigenvalue of a contraction.

Theorem 3.2. *Let H be a contraction of G . Then $\lambda_H \leq \lambda_G$.*

Proof. Without loss of generality, we assume that H is built from G by one step of contracting. Otherwise, we apply mathematical induction. Assume the nodes u and v in $V(G)$ are merged into $u* \in V(H)$. Let f be the harmonic eigenfunction of H , \mathbf{D}_H be the diagonal matrix of H , and e be the unit constant function. Then

$$\mathbf{L}_H f = \lambda \mathbf{D}_H f, \quad f \perp \mathbf{D}_H e.$$

We define the function \tilde{f} on $V(G)$ by

$$\tilde{f}(x) = \begin{cases} f(x), & x \neq u \text{ or } v, \\ f(u*), & x = u \text{ or } v. \end{cases}$$

Since $w(x, u*) = w(x, u) + w(x, v)$, $f \perp \mathbf{D}_H e \implies \tilde{f} \perp \mathbf{D}_G e$,

$$\sum_{x \in N_G(v)} (\tilde{f}(v) - \tilde{f}(x))^2 + \sum_{x \in N_G(u)} (\tilde{f}(u) - \tilde{f}(x))^2 = \sum_{x \in N_H(u*)} (f(u) - f(x))^2,$$

and

$$\tilde{f}^2(u)d_u + \tilde{f}^2(v)d_v = f^2(u*)d_{u*},$$

where d_u and d_v are volumes of u and v in G while d_{u*} is the volume of $u*$ in H . Therefore, we have

$$\frac{\sum_{(x,y) \in E(G)} (\tilde{f}(x) - \tilde{f}(y))^2 w(x, y)}{\sum_{x \in V(G)} \tilde{f}^2(x) d_x} = \frac{\sum_{(x,y) \in E(H)} (f(x) - f(y))^2 w(x, y)}{\sum_{x \in V(H)} f^2(x) d_x} \quad (3.24)$$

and

$$\lambda_H = \frac{\sum_{(x,y) \in E(H)} (f(x) - f(y))^2 w(x, y)}{\sum_{x \in V(H)} f^2(x) d_x}. \quad (3.25)$$

Since

$$\lambda_G = \inf_{\sum f(i)d_i=0} \frac{\sum_{(x,y) \in E(G)} (\tilde{f}(x) - \tilde{f}(y))^2 w(x, y)}{\sum_{x \in V(G)} \tilde{f}^2(x) d_x},$$

by (3.24) and (3.25), we have

$$\lambda_G \leq \frac{\sum_{(x,y) \in E(G)} (\tilde{f}(x) - \tilde{f}(y))^2 w(x, y)}{\sum_{x \in V(G)} \tilde{f}^2(x) d_x} = \lambda_H.$$

The lemma is proved. \square

References

- [1] Bondy, J., Murty, U.: Graph Theory. Springer (2008).
- [2] Chartrand, G.: Introductory Graph Theory. Dover (1985).
- [3] Chung, F.R.: Spectral Graph Theory. CBMS Regional Conference Series in Mathematics, No. 9. AMS (1996).
- [4] Shakhnarovich, G., Darrell, T., Indyk, P. (eds.): Nearest-Neighbor Methods in Learning and Vision, Theory and Practice. MIT (2006).
- [5] Bachmann, C.M., Ainsworth, T.L., Fusina, R.A.: Improved manifold coordinate representations of large-scale hyperspectral scenes. IEEE Trans. Geo. Remote Sensing 44, 2786–2803 (2006).
- [6] Bozkaya, T., Ozsoyoglu, M.: Distance-based indexing for highdimensional metric spaces. In: Proc. ACM SIGMOD, p. 357–368 (1997).
- [7] Friedman, J.H., Bentley, J.L., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM Trans. Math. Softw. 3(3), 209–226 (1977).

- [8] Katayama, N., Satoh, S.: The SR-tree: An index structure for high-dimensional nearest neighbor queries. Proc. ACM SIGMOD p. 369–380 (1997).
- [9] Kim, B.S., Park, S.B.: Fast nearest neighbor finding algorithm based on ordered partition. IEEE Trans. Pattern Anal. Mach. Intell. 8(6), 761–766 (1986).
- [10] Lubiarz, S., Lockwood, P.: Evaluation of fast algorithms for finding the nearest neighbor. Proc. IEEE Int. Conf. Acoust., Speechand Signal Process. 2, 1491–1494 (1997).
- [11] McNames, J.: A fast nearest-neighbor algorithm based on a principal axis search tree. IEEE Trans. Pattern Anal. Mach. Intell. 23(9), 964–976 (2001).
- [12] Yianilos, P.N.: Data structure and algorithms for nearest neighbor search in general metric spaces. Proc. ACM-SIAM Symp. Discr. Algorithms p. 311–321 (1993).
- [13] Chui, C.K.: An Introduction to Wavelets, *Wavelet Analysis and its Applications*, vol. 1. Academic Press, Inc. (1992).
- [14] Chui, C.K., Wang, J.Z.: A cardinal spline approach to wavelets. Proc. Amer. Math. Soc. 113, 785–793 (1991).
- [15] Chui, C.K., Wang, J.Z.: On compactly supported spline wavelets and a duality principle. Trans. Amer. Math. Soc. 330, 903–915 (1992).
- [16] Chui, C.K.: Wavelets: A Mathematical Tool for Signal Analysis. SIAM Monographs on Mathematical Modeling and Computation. Society for Industrial and Applied Mathematics, Philadelphia (1997).
- [17] Chui, C.K., Wang, J.Z.: A general framework of compactly supported splines and wavelets. J. Approx. Theory 71(3), 263–304 (1992).
- [18] Laub, J., Müller, K.R.: Feature discovery in non-metric pairwise data. Journal of Machine Learning Research 5, 801–818 (2004).
- [19] Mahalanobis, P.C.: On the generalised distance in statistics. Proceedings of the National Institute of Sciences of India 2(1), 49–55 (1936).
- [20] De Maesschalck, R., Jouan-Rimbaud, D., Massart, D.: The mahalanobis distance. Chemometrics and Intelligent Laboratory Systems 50, 1–8 (2000).

Chapter 4 Data Models and Structures of Kernels of DR

Abstract In the dimensionality reduction processing, observed data have two different types. In the first type, the data set consists of high-dimensional vectors, which represent the objects of interest. In the second type, the data describe the similarities (or dissimilarities) of objects that cannot be digitized or hidden. The output of a DR processing with an input of the first type is a low-dimensional data set, having the same cardinality as the input and preserving the similarities of the input. When the input is of the second type, the output is a configuration of the input similarities. In Section 1 of this chapter, we discuss the models of input data and the constraints on output data. In Sections 2, we discuss the construction of the kernels in DR methods.

4.1 Data Models in Dimensionality Reduction

4.1.1 Input Data of First Type

As we mentioned in Chapter 1, in many applications, the objects of interest are represented by high-dimensional data, which are point sets in Euclidean spaces. A data set of this type has an extrinsic dimension and an intrinsic dimension. The extrinsic dimension is the dimension of the observed vectors, while the intrinsic dimension is the dimension of the underlying manifold, which the data reside on. If the input of a DR processing is a point set representing objects of interest, this input data set is of the first type. The similarities between the vectors of the first type data have to be “learned” from the data according to the goal of the task.

Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ be an observed data set. In the linear data model, \mathcal{X} is assumed to reside on a hyperplane. Hence,

$$\mathbf{x}_i = \mathbf{a} + \sum_{j=1}^m y_{ij} \mathbf{b}_j, \quad 1 \leq i \leq n, \quad (4.1)$$

where $\mathcal{B} \stackrel{\text{def}}{=} \{\mathbf{b}_1, \dots, \mathbf{b}_m\} \subset \mathbb{R}^D$ is a basis of an m -dimensional subspace of \mathbb{R}^D , and \mathbf{a} is the geometric center of \mathcal{X} . Let \mathcal{B} be orthonormal. Then the DR data set is represented by $\mathcal{Y}_m = \{\mathbf{y}_1, \dots, \mathbf{y}_n\} \subset \mathbb{R}^m$, where $\mathbf{y}_i = [y_{i1}, \dots, y_{im}]'$.

A dimensionality reduction method using linear data model (4.1) is called a linear DR method. In a linear DR method, the dissimilarities between the points of a data set are measured by the Euclidean metric. Let \mathbf{W} denote the Euclidean distance matrix on \mathcal{X} . By the data model (4.1), the Euclidean distance matrix on \mathcal{Y} is equal to \mathbf{W} . Hence, a linear DR method can be described as a linear processing that finds a data set $\mathcal{Y} \subset \mathbb{R}^m$, $m < D$, having the Euclidean distance matrix \mathbf{W} . Note that the data graph $G = [\mathcal{X}, \mathbf{W}]$ in the linear model is completed.

However, in many applications, the similarities between the points of a data set are not described by Euclidean metric, but by other metrics, for example, a Riemannian metric on a manifold. Then the nonlinear data models are applied. In a nonlinear data model, we assume that \mathcal{X} resides on an m -dimensional manifold $M \subset \mathbb{R}^D$, $m < D$. Let g be a parameterization of M denoted by $g = [g^1, \dots, g^D]$, $h = g^{-1} = [h^1, h^2, \dots, h^m]$ be the coordinate mapping from M to U , and the coordinates of $\mathbf{x}_i \in \mathcal{X}$ be denoted by \mathbf{y}_i . Then the data set \mathcal{X} is modeled by

$$\mathbf{x}_i = g(\mathbf{y}_i), \quad i = 1, \dots, n, \quad (4.2)$$

which can also be written as

$$\mathbf{y}_i = h(\mathbf{x}_i), \quad i = 1, \dots, n. \quad (4.3)$$

The data set $\mathcal{Y}_m = \{\mathbf{y}_1, \dots, \mathbf{y}_n\} \subset \mathbb{R}^m$ is called a (nonlinear) dimensionality reduction of \mathcal{X} (with the intrinsic dimension).

In general, the observed data \mathcal{X} is contaminated by noise. Let the noise on the observed vector \mathbf{x}_i be denoted by $\boldsymbol{\varepsilon}_i$. The noisy observed data is modeled as

$$\mathbf{x}_i^* = g(\mathbf{y}_i) + \boldsymbol{\varepsilon}_i, \quad i = 1, \dots, n. \quad (4.4)$$

The noise on \mathcal{X} will be transferred to the DR data \mathcal{Y}_m in the DR processing. We denote the noise on the DR vector \mathbf{y}_i by $\boldsymbol{\eta}_i$. Applying the Taylor's formula to (4.3), we have

$$\begin{aligned} \mathbf{y}_i + \boldsymbol{\eta}_i &= h(\mathbf{x}_i + \boldsymbol{\varepsilon}_i) \\ &= h(\mathbf{x}_i) + dh_{\mathbf{x}_i} \boldsymbol{\varepsilon}_i + O(\|\boldsymbol{\varepsilon}_i\|^2). \end{aligned}$$

If the variation of the noise is small, omitting the quadratic term in the equation above, we obtain

$$E(\|\boldsymbol{\eta}_i\|^2) \leq \|dh_{\mathbf{x}_i}\|^2 E(\|\boldsymbol{\varepsilon}_i\|^2), \quad i = 1, \dots, n. \quad (4.5)$$

The equation (4.5) is a pointwise estimation. Replacing $\|dh_{\mathbf{x}_i}\|$ by the global norm $\|dh(\mathcal{X})\| \stackrel{\text{def}}{=} \max_{\mathbf{x}_i \in \mathcal{X}} \|dh_{\mathbf{x}_i}\|$, we obtain a global estimation

$$\sum_{i=1}^n E(\|\boldsymbol{\eta}_i\|^2) \leq \|dh(\mathcal{X})\|^2 \sum_{i=1}^n E(\|\boldsymbol{\varepsilon}_i\|^2).$$

Note that

$$\|dh_{\mathbf{x}_i}\|^2 = \|\mathbf{G}_i\|,$$

where $\mathbf{G}_i = (dh_{\mathbf{x}_i})' dh_{\mathbf{x}_i}$. The noise estimate formula (4.5) is valid for all DR methods, only $\|\mathbf{G}_i\|$ alters. Since the noise estimate is not our main concern in this book, later we shall focus our discussion on the noiseless data model (4.2), or the equivalent model (4.3).

4.1.2 Input Data of Second Type

In some applications, the objects are not possible or not necessary to be digitized. Many objects come from social sciences, biology, economy, and other areas are in this category. Let V be a set of objects of this type. Assume that the similarities between the objects in V (or a part of them) are measured. Then a DR method is employed to find a *virtual data set* in a certain Euclidean space, say, $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\} \subset \mathbb{R}^m$, such that the Euclidean metric on \mathcal{Y} preserves the similarities on V . We shall assume an input is of the second type if it is a similarity matrix, say,

$$\mathbf{W} = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ w_{21} & \cdots & w_{2n} \\ \vdots & & \vdots \\ w_{n1} & \cdots & w_{nn} \end{pmatrix},$$

where the entry w_{ij} measures the similarity between the i th and j th objects of V .

Remark 4.1. In DR, the term “similarity” is used to stand for either similarity or dissimilarity. We shall call a similarity matrix \mathbf{W} of covariance-type if large values of the entries in \mathbf{W} describe the similarities, and regard it as distance-type if small values describe the similarities. Since in practice distances (of certain types) are commonly used to measure similarities, later we shall assume that the similarity matrix is of distance-type.

Assume that the input is of the second type. The data model in a DR processing now is

$$w_{ij} = d_2(\mathbf{y}_i, \mathbf{y}_j), \quad \mathbf{y}_i \in \mathbb{R}^m, \quad w_{ij} \neq 0, \quad 1 < i, j < n, \quad (4.6)$$

The data set \mathcal{Y} in (4.6) is called a configuration of \mathbf{W} . It is clear that \mathcal{Y} is not unique. If there is a set $\mathcal{Y} \subset \mathbb{R}^m$ satisfying (4.6), then for each $k \geq m$, there is a set $\mathcal{Y}_k \subset \mathbb{R}^k$ which also satisfies (4.6). The minimal k is called the intrinsic configuration dimension of \mathbf{W} . The matrix \mathbf{W} in (4.6) defines a data graph $G_V = [V, \mathbf{W}]$. Some DR methods can be applied to finding a configuration $\mathcal{Y} \subset \mathbb{R}^m$ such that its Euclidean distance matrix approximates \mathbf{W} .

If the input matrix \mathbf{W} is a Euclidean distance matrix, then we call the data model (4.6) linear, otherwise it is called nonlinear. In a linear model, \mathbf{W} is dense. The nonlinear model is often obtained in the following situation: For each object $v \in V$, only the distances between v and its a few nearest neighbors are measured. Then, the input matrix \mathbf{W} is sparse, in which unavailable similarities are set at zero on the set V . The noise may impact the measurements of the entries of \mathbf{W} .

Remark 4.2. In some applications, the similarities between the objects are measured by several metrics. For example, let the objects on the set V be the major cities in the US. We use three different metrics, (1) flight mileage, (2) price of flight ticket, and (3) flight time, to measure the “flight distances” between cities. We also assume that these distances are measured only if a non-stop flight exists between the cities. Thus, we obtain an array of similarity matrices on V . In this case, we need from these matrices to construct a composed similarity matrix as the input data of the model (4.6).

4.1.3 Constraints on Output Data

In most dimensionality reduction problems, the output data is not required to have the intrinsic dimension, say, m , but a target dimension d that is lower than the intrinsic one, $d < m$. We shall point out that using a target dimension instead of the intrinsic dimension does not change DR models much. Let us first consider input data of the first type. When a manifold coordinate mapping h^1, h^2, \dots, h^m in (4.3) is properly selected, these functions are the feature functions of data \mathcal{X} in terms of *machine learning*. The degrees of importance of these features are not the same. Some features are more important than others in a certain application. When we apply a DR processing to the application, we only need to find d most important feature functions, say, the functions h^1, \dots, h^d , if they are features of interest in the application. Then we shall set the target dimension at d , not m .

The DR data set $\mathcal{Y} \subset \mathbb{R}^d$ is not unique. Shifting and affine transformation do not change the data geometry much. Hence, for a certainty, we can require that \mathcal{Y} is centered, i.e., $\sum_{i=1}^n \mathbf{y}_i = 0$, and the y -coordinates are orthogonal. Let \mathbf{Y} be the data matrix for \mathcal{Y} . These constraints can be written as

$$\mathbf{Y}\mathbf{1} = 0, \quad (4.7)$$

$$\mathbf{Y}\mathbf{Y}' = \mathbf{I}, \quad (4.8)$$

where $\mathbf{1}$ denotes the vector with the unit entries. In some applications such as classification, the output data need to be scaled for better separation, then the coordinates can be rescaled. For instance, let $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, \dots, \sigma_d)$ be a diagonal matrix with positive diagonal entries. Then the output data matrix \mathbf{Y} can be scaled to $\boldsymbol{\Sigma}\mathbf{Y}$. This means that the constraint (4.8) is modified to

$$\mathbf{Y}\mathbf{Y}' = \boldsymbol{\Sigma}^2. \quad (4.9)$$

We shall call the first constraint (4.7) *centralization constraint* and the second one (4.8) *orthogonality constraint*. When the input data is of the second type, the discussion on the output is similar. First consideration is about the dimension of the output data set. The intrinsic configuration dimension of the input similarity matrix \mathbf{W} may be quite high. When the output configuration \mathcal{Y} is used for data visualization, The dimension d of the points in \mathcal{Y} is often equal to 2 or 3. Therefore, among all d -dimensional configuration sets, the output data \mathcal{Y} is obtained by maximizing the variance with respect to the input similarity matrix. In this case, the constraints (4.7) and (4.9) are applied.

4.1.4 Consistence of Data Graph

Assume that the input data \mathcal{X} is of the first type. Then, in DR, we construct a distance-type weight matrix \mathbf{W} on \mathcal{X} to create a data graph $G = [\mathcal{X}, \mathbf{W}]$, which describes the geometric structure of data. In nonlinear model, we construct the data graph using either k -neighborhood or ε -neighborhood as discussed in Section 3.1. Let \mathbf{A} be the adjacency matrix of G and set $N(i) = \{j : \mathbf{A}(i, j) = 1\}$. Then the *graph neighborhood* of \mathbf{x}_i is the set

$$O(i) = \{\mathbf{x}_j, \quad j \in N(i)\}.$$

A well-defined graph G is consistent with the geometry of M . More precisely, let U_i be the neighborhood of \mathbf{x}_i on the manifold M , which is simply called *manifold neighborhood*. Then the graph neighborhood O_i must be a subset of the manifold neighborhood U_i and on the other hand, O_i must be a “frame” of U_i in the sense that each point on U_i can be (approximately) represented as a linear combination of O_i . However, because the manifold M is underlying, it is hard to check whether the graph is consistent with M . Hence, we set some weaker consistent conditions on the graph G :

Connection consistence. The graph G is connected if and only if M is connected.

Local dimension consistence. $\dim(\text{span}(O(i))) \geq d$,

where d is the target dimension. The first condition ensures that the graph preserves the connection of the manifold, and the second condition ensures that the graph does not lose local data features. For instance, if the second condition is not satisfied, then there is a neighborhood O_i that spans only an s -dimensional subspace ($s < d$), then the d -dimensional local features (around \mathbf{x}_i) will be lost in the DR processing. We shall call these conditions *the graph (neighborhood) consistence conditions*. In the construction of k -neighborhood (or ε -neighborhood), a suitable large k (or epsilon) can meet the need of the second condition. We now discuss the first condition.

4.1.5 Robust Graph Connection Algorithm

In this subsection, we first introduce an algorithm to check whether a graph is connected. Then a robust graph connection algorithm is developed to modify a disconnected graph to a connected one by enlarging graph neighbors.

Assume that a data set is a sample set of a connected manifold. Therefore, its data graph must be connected. In Section 3.3, we learn that a graph is connected if and only if the Laplacian of the graph has a simple 0-eigenvalue. But, for a large-dimensional matrix, finding the multiplicity of an eigenvalue is computing expensive and numerically unstable. The presented algorithm checks the graph connection based on the following idea: If a graph is disconnected, then no paths have the length equal to the cardinality of the node set. The algorithm is fast and stable. The following is its Matlab code.

```

function tf = isconnected(A)
% Determine whether a graph is connected.
% If a graph is not connected, then no
% path through all nodes, i.e., length
% of any path is less than length(A).
% Syntax: tf = isconnected(A)
%
% Input: A - adjacency matrix of graph.
% Output: (Boolean) TF - true or false.
%
% Initialize output
tf = false;
% get the length of A.
n = size(A,1);
% Initial the neighbor of Node 1.
olds = find(A(1,:));
% Initial the first path.
nbrs = unique([1, olds]);
% Assume Node 1 is not isolate.
if length(nbrs)>1
% Make path starting from Node 1.
%When Node 1 has only one neighbor, make olds

```

```
%have at least 2 nodes so that
%sum(A(olds,:)) is a vector.
if length(olds)==1
    olds = nbrs;
    nbrs = find(sum(A(olds,:)));
end
while length(nbrs)>length(olds)
    % count old path length
    olds = nbrs;
    % update path
    nbrs = find(sum(A(olds,:)));
end
end
if length(nbrs)==n
    tf=true;
end
```

The robust connection algorithm adopts the previous algorithm `isconnected.m` to check the graph connection. If the graph is found disconnected, it enlarges neighborhood iteratively till the graph is connected. The input of the algorithm is the distance-type dissimilarity matrix. The outputs are the weight matrix and the adjacency matrix for the updated connected graph.

```
function [ND2, A] = make_connectedgraph(S,options)
% Make connected graph for local similarities.
% [ND2, A] = MAKE_CONNECTEDGRAPH(S, OPTIONS)
% INPUTS:
%   S: (n x n) a distance-type similarity matrix,
%       in which X(j,i) is dissimilarity of i,j.
%   OPTIONS:
%     .epsilon(double) for epsilon-neighbors.
%     .k (integer or n-vector) for k-neighbors.
%     .verb (Boolean) if =1, display the comments.
% OUTPUTS:
%   ND2 (double,sparse): sparse matrix, in which
%   each row contains only the similarities
%   from the i-th point to its neighbors.
%   ND2(i,j)=0, if point j is not a neighbor of
%   the i-th point.
%   Otherwise, ND2(i,j) = S(i,j).
%   A (Boolean, sparse): the adjacency matrix.
%   A(i,j) = 1, if x_j is a neighbor of x_i.
%   Otherwise, A(i,j) = 0.
%
% Call: isconnected.m
%
% Example.
% X = rand(200,400);
% S = square_distance(X);
```

```

% options.k = 5;
% options.verb = 1;
% [ND2,A]= make_connectedgraph(S, options);
%
% Initialize options.
options.null=0;
n = length(S);
emode = false;
if isfield(options, 'verb')
    verb = options.verb;
else
    verb = 1;
end
if isfield(options, 'epsilon')
    epsilon = options.epsilon;
    emode = true;
elseif isfield(options, 'k')
    k = options.k;
    %checkthe validity of k
    if (numel(k) ~= 1)&&(numel(k) ~= n)
        error('k must be integer or n-vector');
    elseif numel(k)==1
        k=k*ones(1,n);
    else
        k=k(:);
    end
else
    k = 3*ones(1,n);
end
% Construct neighborhood and compute
% local weights for nearest neighbors.
if emode % Compute eps-neighborhood.
    if verb
        disp('- computing eps-neighborhood.');
    end
    S(S> epsilon) = 0;
    ND2 = sparse(S);
    A=(ND2>0);

    % check connection of graph, if not
    % update neighbors to force it connected
    while ~isconnected(A)
        epsilon = 1.1*epsilon;
        S(S> epsilon) = 0;
        ND2 = sparse(S);
        A=(ND2>0);
    end
else % Compute k-neighborhood.
    if verb
        disp('- computing k-neighborhood.');
    end

```

```

end
ND2 = sparse(n,n);
[sortS, indexS] = sort(S);
for i = 1:n
    nbri = indexS(2:k(i)+1,i);
    ND2(nbri,i) = sortS(2:k(i)+1,i);
end
A = (ND2>0);
% check connection of graph, if not
% update neighbors to force it connected
while ~isconnected(A)
    for i = 1:n
        addnbri = indexS(k(i)+1,i);
        ND2(addnbri,i) = sortS(k(i)+1,i);
    end
    ND2 = sparse(ND2);
    A = (ND2>0);
end
end

```

4.2 Constructions of DR Kernels

The key step of the spectral approach to DR is the construction of a DR kernel. A DR kernel is a psd matrix whose spectral decomposition yields the DR data. In this section we shall briefly discuss the types of DR kernels and the framework of the construction. All detailed discussions about DR kernel constructions are in the context of the remaining parts of the book.

4.2.1 DR Kernels of Linear Methods

In a linear method, the DR data is a projection of the original data. Hence, its DR kernel is the matrix representation of the project operator. Two main (deterministic) linear methods are principal component analysis (PCA) and classical multidimensional scaling (LMDS). In Chapter 5, we shall introduce PCA in detail. Here, we only outline the method from the kernel point of view. In PCA, the input data is \mathcal{X} . For convenience, we assume that the original data $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ is already centered: $\sum_{i=1}^n \mathbf{x}_i = 0$. PCA projects \mathbf{x}_i onto a d -dimensional subspace that minimizes the approximation error. Let the projection matrix be denoted by \mathbf{K} of rank d . Then \mathbf{K} is the solution of the least-square problem:

$$\mathbf{K} = \arg \min_{\mathbf{K} \in S} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{K}\mathbf{x}_i\|^2. \quad (4.10)$$

Let the optimal projection matrix, as the solution of (4.10), be factorized as $\mathbf{K} = \mathbf{U}\mathbf{U}^T$, where the columns of $\mathbf{U} \in \mathfrak{O}_{D \times d}$ form an o.n. basis of a d -dimensional subspace in \mathbb{R}^D . Then the d -dimensional DR vectors are given by

$$\mathbf{y}_i = \mathbf{U}^T \mathbf{x}_i, \quad i = 1, 2, \dots, n.$$

Hence, the kernel \mathbf{K} of PCA is the covariance matrix $\mathbf{K} = \frac{1}{n} \mathbf{X} \mathbf{X}'$.

Classical multidimensional scaling CMDS will be discussed in Chapter 6. The input of CMDS can be either an object vector set \mathcal{X} or an $n \times n$ Euclidean distance matrix D , which describes the similarities between the objects in a set V of n objects. In Chapter 6, we shall learn how to convert a Euclidean distance matrix to a psd matrix \mathbf{K} , which represents the Gram matrix \mathbf{G} of a centered data set. Then the kernel of CMDS is the centering Gram matrix \mathbf{G} . The output \mathcal{Y} is obtained from the scaled leading eigenvectors of \mathbf{K} . When the input data of CMDS is a data set \mathcal{X} , CMDS is essentially identified with PCA.

4.2.2 DR Kernels of Nonlinear Methods

In a nonlinear DR model, the output data are the manifold coordinate representation or feature vectors of the original data. Because the underlying manifold M in the data model (4.3) is unknown, we cannot directly compute the manifold coordinates of input vectors. Then we first construct a DR kernel, from which the coordinates of \mathcal{X} can be “learned”. The DR kernel is constructed based on the data graph. For convenience of discussion, in the following, we assume that the target dimension is the same as the intrinsic dimension. This assumption will not lose the generality since, when the target dimension is used in DR processing, the only difference is the number of eigenvectors selected from the spectral decomposition of the DR kernel. Technically, there are two approaches to the construction of nonlinear DR kernels, *distance approach* and *similarity weight approach*.

We now discuss the distance approach to nonlinear DR. Let $O(i)$ be the manifold neighborhood of \mathbf{x}_i and h be the coordinate mapping on M . The local approximation of (4.3) is

$$\mathbf{y}_j = dh_{\bar{\mathbf{x}}}(\mathbf{x}_j - \mathbf{x}_i), \quad j \in N(i), \quad (4.11)$$

which yields

$$\mathbf{y}_j - \mathbf{y}_k = dh_{\mathbf{x}_i}(\mathbf{x}_j - \mathbf{x}_k), \quad j, k \in N(i).$$

Let $\mathbf{G}_i = (dh_{\mathbf{x}_i})' dh_{\mathbf{x}_i}$ and write

$$m_i(j, k) \stackrel{\text{def}}{=} (\mathbf{x}_j - \mathbf{x}_k)' \mathbf{G}_i (\mathbf{x}_j - \mathbf{x}_k). \quad (4.12)$$

Then

$$\|\mathbf{y}_j - \mathbf{y}_k\|^2 = m_i(j, k), \quad 1 \leq j, k \leq n. \quad (4.13)$$

If h is an isometric isomorphism, then \mathbf{G}_i is a local distance-preserving operator such that

$$m_i(j, k) = \|\mathbf{x}_j - \mathbf{x}_k\|^2, \quad j, k \in N(i). \quad (4.14)$$

In the distance approach, the local relation (4.13) is used to construct a global metric $\mathbf{S} = [s_{jk}]_{j,k=1}^n$ such that $s_{jk} = m_i(j, k)$. Then the DR set \mathcal{Y} is obtained from the global relation (4.13). The matrix \mathbf{S} usually is not psd. To construct a spd kernel \mathbf{K} , we define the kernel distance by

$$d_{\mathbf{K}}(i, j) = \sqrt{k_{ii} + k_{jj} - 2k_{ij}}.$$

Then, we construct \mathbf{K} using the relation $s_{jk} = d_{\mathbf{K}}^2(i, j)$. Correspondingly, the output data is yielded from the first d -leading eigenvectors of \mathbf{K} .

The similarity weight approach to nonlinear DR preserves the weights of a point with respect to its neighbors. Let the tangent hyperplane of M at \mathbf{x}_i be denoted by TH_i and the projection of \mathbf{x}_j onto TH_i be denoted by \hat{x}_j . When $j \in N(i)$, $\mathbf{x}_j \approx \hat{x}_j$. Applying the linear approximation to the data model (4.2), we have

$$dg_{\mathbf{y}_i} \mathbf{y}_j = \hat{x}_j - \hat{x}_i \approx \mathbf{x}_j - \mathbf{x}_i, \quad j \in N(i). \quad (4.15)$$

Assume that $|N(i)| = k > d$, then there is a k -dimensional vector $\mathbf{w}_i = [w_{ji}], j \in N(i)$ such that

$$\sum_{j \in N(i)} w_{ji} \hat{x}_j = \hat{x}_i, \quad (4.16)$$

with the constraint $\sum_{j \in N(i)} w_{ji} = 1$. The numbers $w_{ji}, j \in N(i)$, in (4.16), are called the similarity weights of \mathbf{x}_i with respect to its neighbors, for they describe the similarities between \mathbf{x}_i to its neighbors. By (4.15), we obtain

$$\sum_{j \in N(i)} w_{ji} \mathbf{y}_j = 0, \quad 1 \leq i \leq n. \quad (4.17)$$

Extending the local relation (4.17) to a global one, we obtain a global relation

$$(\mathbf{I} - \mathbf{W}) \mathbf{Y}' = 0, \quad 1 \leq i \leq n, \quad (4.18)$$

where \mathbf{W} is an $n \times n$ weight matrix and \mathbf{Y} is the data matrix of \mathcal{Y} . To eliminate the impact of noise and modeling errors, we apply the optimization technique, obtaining the minimization model:

$$\mathbf{Y} = \arg \min_{\mathbf{Y} \in \mathfrak{D}_{d,n}} \text{tr}(\mathbf{Y} (\mathbf{I} - \mathbf{W})' (\mathbf{I} - \mathbf{W}) \mathbf{Y}') \quad (4.19)$$

with the constraints (4.7) and (4.8) (or (4.9)). Hence, the DR kernel is $\mathbf{K} = \mathbf{W}' \mathbf{W}$ and key of this approach is to construct the weight matrix

\mathbf{W} . The output of this approach is yielded from the 2nd to $(d+1)$ th bottom eigenvectors of \mathbf{K} , recalling that the bottom eigenvector is $\frac{1}{\sqrt{n}}\mathbf{1}$. All local linear embedding methods and their variations such as LLE, LTSA, Leigs, and HLLE create DR kernels of this type.

4.2.3 Conclusion

The distance approach establishes the relation between the Euclidean metric on the DR data set and a non-Euclidean metric, such as the Riemannian metric on manifold or local Euclidean metric on the original data. The DR kernel is constructed so that the kernel distance preserves the defined non-Euclidean distance on the original data. The kernels of this type are often dense and the DR data is obtained from its leading eigenvectors. Using a kernel of this type in a DR processing, the energy loss can be estimated from the kernel spectrum. The similarity weight approach constructs a kernel that preserves the locally linear relations among the points in a neighborhood of the data. The kernel is sparse, and the DR data is derived from a numerically null space of the kernel. The spectral decompositions for the kernels of this type are faster. But some methods of this type do not guarantee the local distance preservation, and do not provide the estimates of the energy losses in the DR processing either. We illustrate these two approaches in Fig. 4.1 and Fig. 4.2.



Fig. 4.1 Nonlinear Dimensionality Reduction Approach 1



Fig. 4.2 Nonlinear Dimensionality Reduction Approach 2

There is no essential difference between these two approaches since both of them adopt local geometric structure of the data set to construct the kernels. Due to the finiteness of the data sets in DR, each data set can be embedded in infinitely many different manifolds, depending on the definitions of the neighborhood systems on the data. Unfortunately, the question as to “which manifold is the best for the given data set” is hard to be answered mathematically. Similarly, it is hard to set quantitative evaluation rules for the quality of DR methods, unless some prior knowledge of the underlying manifold is known. The reviews of spectral approach to DR are referred to

[1–3]. A study of the relations among various nonlinear DR methods is referred to [4].

References

- [1] Carreira-Perpiñán, M.Á.: A review of dimension reduction techniques. Tech. Rep. CS-96-09, Dept. of Computer Science, University of Sheffield, UK (1996).
- [2] Lu, F., Keleş, S., Lin, Y., Wright, S., Wahba, G.: Kernel regularization and dimension reduct. Tech. rep., Department of Statistics, University of Wisconsin (2006).
- [3] Saul, L.K., Weinberger, K.Q., Sha, F., Ham, J., Lee, D.D.: Spectral methods for dimensionality reduction. In: B. Schölkopf, O. Chapelle, A. Zien (eds.) *Semisupervised learning*. MIT Press (2005).
- [4] Xiao, L., Sun, J., Boyd, S.P.: A duality view of spectral methods for dimensionality reduction. In: W.W. Cohen, A. Moore (eds.) *Machine Learning: Proceedings of the Twenty-Third International Conference, ACM International Conference Proceeding Series*, vol. 148, pp. 1041–1048. ICML, Pittsburgh, Pennsylvania, USA (2006).

Part II

Linear Dimensionality Reduction

Chapter 5 Principal Component Analysis

Abstract Among linear DR methods, principal component analysis (PCA) perhaps is the most important one. In linear DR, the dissimilarity of two points in a data set is defined by the Euclidean distance between them, and correspondingly, the similarity is described by their inner product. Linear DR methods adopt the global neighborhood system: the neighbors of a point in the data set consist of all of other points. Let the original data set be $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ and the DR data set of \mathcal{X} be a d -dimensional set \mathcal{Y} . Under the Euclidean measure, PCA finds a linear projection $T : \mathbb{R}^D \rightarrow \mathbb{R}^d$ so that the DR data $\mathcal{Y} = T(\mathcal{X})$ maximize the data energy. PCA is widely used in many applications. The present chapter is organized as follows. In Section 5.1, we discuss the description of PCA. In Section 5.2, we present the PCA algorithms. Some real-world applications of PCA are introduced in Section 5.3.

5.1 Description of Principal Component Analysis

The principal component analysis (PCA) method was perhaps first invented by Pearson in the study of the best fit line and space for given data [1]. A comprehensive review is referred to the book [2]. PCA is the most important linear DR method. It can be described from two different points of view: geometric and statistical. Let us first take the geometric point of view.

5.1.1 Geometric Description of PCA

We already know that shift does not change data geometry. Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ be a given data set and $\mathbf{a} \in \mathbb{R}^D$. We denote by $\mathcal{X}_{\mathbf{a}}$, or $\mathcal{X} - \mathbf{a}$, the \mathbf{a} -shift of \mathcal{X} :

$$\mathcal{X}_{\mathbf{a}} = \{\mathbf{x} - \mathbf{a} \in \mathbb{R}^D : \mathbf{x} \in \mathcal{X}\}.$$

Since any \mathbf{a} -shift of \mathcal{X} is geometrically equivalent to \mathcal{X} , in PCA we always shift \mathcal{X} by its geometric center

$$\bar{\mathbf{x}} = \sum_{i=1}^n \mathbf{x}_i,$$

so that the *centered data* $\hat{\mathcal{X}} = \mathcal{X}_{\bar{\mathbf{x}}}$ is to be used. Let the energy of \mathcal{X} be defined by

$$\mathcal{E}(\mathcal{X}) = \sum_{i=1}^n \|\mathbf{x}_i\|^2 = \|\mathbf{X}\|_F^2,$$

where $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_n]$ is the data matrix and $\|\mathbf{X}\|_F$ is the Frobenius norm of \mathbf{X} . Then the centered data $\hat{\mathcal{X}}$ has the minimal energy among all shifts of \mathcal{X} :

$$\mathcal{E}(\hat{\mathcal{X}}) = \min_{\mathbf{a} \in \mathbb{R}^D} \mathcal{E}(\mathcal{X}_{\mathbf{a}}).$$

For simplicity, in this chapter, we shall assume that the original data set \mathcal{X} is already centered, i.e., $\hat{\mathcal{X}} = \mathcal{X}$.

Roughly speaking, PCA is the method that finds the subspace where the given data set concentrates. To describe the *concentration* of \mathcal{X} , we consider the following maximization problem: On which line does the projection of \mathcal{X} preserve the greatest energy? To answer the question, we denote the line by S_1 . Since the data \mathcal{X} is centered, the line S_1 must be through the origin, i.e., it is a 1-dimensional subspace of \mathbb{R}^D . We denote the direction of S_1 by the unit vector \mathbf{v}_1 , the o.g. project operator $\mathbb{R}^D \rightarrow S_1$ by $T_{\mathbf{v}_1}$, and the set of all directions in \mathbb{R}^D by the unit sphere $\mathbb{S}^{D-1} \subset \mathbb{R}^D$. Then the vector \mathbf{v}_1 is the solution of the following maximization problem:

$$\mathbf{v}_1 = \arg \max_{\mathbf{a} \in \mathbb{S}^{D-1}} \mathcal{E}(\mathbf{T}_{\mathbf{a}}(\mathcal{X})). \quad (5.1)$$

We denote

$$\mathbf{Y}_1 = [y_{1,1}, \dots, y_{1,n}]' = \mathbf{T}_{\mathbf{a}}(\mathcal{X}) = [\mathbf{T}_{\mathbf{a}}(\mathbf{x}_1), \dots, \mathbf{T}_{\mathbf{a}}(\mathbf{x}_n)]' \in \mathbb{R}^n.$$

We shall call $\mathbf{v}_1 \in \mathbb{S}^{D-1}$ the *first principal direction*, and \mathbf{Y}_1 is the *first principal component* of \mathcal{X} . By mathematical induction, we define the successive principal directions and corresponding principal components as follows. Assume that the first $s-1$ principal directions $\mathcal{V}_{s-1} = \{\mathbf{v}_1, \dots, \mathbf{v}_{s-1}\} \subset \mathbb{S}^{D-1}$ and the corresponding $d-1$ principal components $\{Y_1, \dots, Y_{s-1}\} \subset \mathbb{R}^n$ of \mathcal{X} are well defined, where \mathcal{V}_{s-1} is an o.n. system in \mathbb{R}^D . Define

$$\mathcal{X}_{s-1} = \left\{ \mathbf{x}_i - \sum_{j=1}^{s-1} \mathbf{v}_j y_{j,i}, \quad 1 \leq i \leq n \right\},$$

and

$$S_{s-1} = \text{span}(\mathbf{v}_1, \dots, \mathbf{v}_{s-1}), \quad s = 2, 3, \dots.$$

Then the s th principal direction is defined by

$$\mathbf{v}_s = \arg \max_{\mathbf{a} \in S_{s-1}^\perp \cap \mathbb{S}^{D-1}} \mathcal{E}(\mathbf{T}_{\mathbf{a}}(\mathcal{X}_{s-1})), \quad s = 2, 3, \dots, \quad (5.2)$$

and the s th principal component of \mathcal{X} is $Y_s = T_{\mathbf{v}_s}(\mathcal{X}_{s-1})$. Thus, the *concentration* can be mathematically explained by principal directions and principal components.

Remark 5.1. The maximization problems (5.1) and (5.2) can be converted to the following minimization problems:

$$\mathbf{v}_1 = \arg \min_{\mathbf{a} \in \mathbb{S}^{D-1}} \mathcal{E}((\mathbf{I} - \mathbf{T}_{\mathbf{a}})(\mathcal{X})), \quad (5.3)$$

and

$$\mathbf{v}_s = \arg \min_{\mathbf{a} \in S_{s-1}^\perp \cap \mathbb{R}^D} \mathcal{E}((\mathbf{I} - \mathbf{T}_{\mathbf{a}})(\mathcal{X}_{s-1})). \quad (5.4)$$

Let us make the induction (5.2) up to d . The induction indicates that (1) $\mathcal{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_d\}$ is an o.n. basis of the d -dimensional subspace $S_d \subset \mathbb{R}^D$, and (2)

$$\mathbf{y}_i = \sum_{j=1}^d \mathbf{v}_j y_{j,i} \quad (5.5)$$

is the o.g. projection of \mathbf{x}_i on S_d such that the data set $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\}$ preserves the maximal energy of \mathcal{X} among all d -dimensional data. Therefore, we conclude that PCA is the method to find the d -dimensional projection of \mathcal{X} that maximizes data energy. We call \mathcal{Y} a d dimensionality reduction of \mathcal{X} .

The solutions \mathcal{V} and \mathcal{Y} can be obtained with the aid of the best d -rank approximation for matrices (P 11.5.5 in [3]).

Theorem 5.1 (Mirsky (1960), Eckart and Young (1936)). Let the singular value decomposition (SVD) of a matrix $\mathbf{X} \in \mathfrak{M}_{D,n}(r)$ be

$$\mathbf{X} = \mathbf{V} \boldsymbol{\Sigma} \mathbf{U}' = \sum_{i=1}^r \sigma_i \mathbf{v}_i (\mathbf{u}_i)', \quad (5.6)$$

where $\mathbf{U} = [\mathbf{u}_1 \ \dots \ \mathbf{u}_r] \in \mathfrak{O}_{n,r}$, $\mathbf{V} = [\mathbf{v}_1 \ \dots \ \mathbf{v}_r] \in \mathfrak{O}_{D,r}$, and $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, \dots, \sigma_r)$ is a diagonal matrix with $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r (> 0)$. Let

$$\mathbf{X}_* = \sum_{i=1}^d \sigma_i \mathbf{v}_i (\mathbf{u}_i)', \quad d \leq r. \quad (5.7)$$

Then $\mathbf{X}_* \in \mathfrak{M}_{D,n}(d)$ in (5.7) is the best d -rank approximation of \mathbf{X} under the Frobenius norm, namely,

$$\|\mathbf{X} - \mathbf{X}_*\|_F = \min_{\mathbf{B} \in \mathfrak{M}_{D,n}(d)} \|\mathbf{X} - \mathbf{B}\|_F,$$

with the error given by

$$\|\mathbf{X} - \mathbf{X}_*\|_F = \sqrt{\sum_{l=k+1}^r \sigma_l^2}.$$

From Theorem 5.1, we immediately obtain:

Theorem 5.2. Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ be given (centered) data and the SVD of \mathbf{X} be given in (5.6). Write $\mathcal{V}_d = \{\mathbf{v}_1, \dots, \mathbf{v}_d\} \subset \mathbb{R}^D$, $\Sigma_d = \text{diag}(\sigma_1, \dots, \sigma_d)$, and $\mathbf{U}_d = [\mathbf{u}_1 \cdots \mathbf{u}_d]$. Then the d principal directions of \mathcal{X} are \mathcal{V} , the \mathcal{V} -coordinates of the d principal components of \mathcal{X} are the row vector of the matrix

$$\mathbf{Y} (\stackrel{\text{def}}{=} [\mathbf{y}_1 \cdots \mathbf{y}_n]) = \Sigma_d (\mathbf{U}_d)', \quad (5.8)$$

and the DR data set is $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\}$.

Proof. Let $r = \text{rank}(\mathbf{X})$. Without loss of generality, we assume that $d \leq r$. For $1 \leq s \leq d$, write $\mathbf{V}_s = [\mathbf{v}_1 \cdots \mathbf{v}_s]$, $\Sigma_s = \text{diag}(\sigma_1, \dots, \sigma_s)$, $\mathbf{U}_s = [\mathbf{u}_1 \cdots \mathbf{u}_s]$, and $\mathbf{Y}_s = \Sigma_s \mathbf{U}'_s$. It is clear that \mathbf{Y}_s consists of the first s rows of \mathbf{Y} . The matrix $\mathbf{B}_s \stackrel{\text{def}}{=} \mathbf{V}_s \mathbf{Y}_s$ is in $\mathfrak{M}_{D,n}(s)$. By Theorem 5.1,

$$\|\mathbf{X} - \mathbf{B}_s\|_F = \min_{\mathbf{B} \in \mathfrak{M}_{D,n}(s)} \|\mathbf{X} - \mathbf{B}\|_F, \quad 1 \leq s \leq d,$$

which implies that $\{\mathbf{v}_1, \dots, \mathbf{v}_d\} \subset \mathbb{R}^D$ are successive principal directions of \mathcal{X} and the rows of \mathbf{Y} are the corresponding d principal components of \mathcal{X} . The theorem is proved. \square

By Theorem 5.2, the DR data can be directly obtained from the SVD of data matrix \mathbf{X} using the formula (5.8). People often compute \mathcal{Y} using the spectral decomposition of the covariance matrix of \mathbf{X} , which is defined by

$$\mathbf{F} = \frac{1}{n} \mathbf{X} \mathbf{X}'. \quad (5.9)$$

Let the spectral decomposition of F be

$$\mathbf{F} = \mathbf{V} \Lambda \mathbf{V}'.$$

Then the matrix \mathbf{V} is the same as in (5.6). Hence, we can extract the submatrix \mathbf{V}_d from \mathbf{V} . Since the columns of \mathbf{V}_d are the o.n. basis of the best approximative subspace S_d , the o.n. projection of $\mathbf{x}_i \in \mathcal{X}$ onto S_d is $\mathbf{V}' \mathbf{x}_i$, which yields

$$\mathbf{Y} = (\mathbf{V}_d)' \mathbf{X}. \quad (5.10)$$

In many applications, the number of points n is much greater than the dimension D : $D \ll n$. Therefore, the covariance matrix \mathbf{F} has a relatively lower dimension, say, $D \times D$, so that the spectral decomposition of \mathbf{F} is quite fast.

5.1.2 Statistical Description of PCA

Next, let us take the statistical point of view for PCA. Let $\mathcal{X} \subset \mathbb{R}^D$ be a given data set and $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_n]$ be its matrix representation. We now consider \mathcal{X} as the random vector

$$\mathbf{X} = [X_1, \dots, X_D]' \in \mathbb{R}^D,$$

for which each column vector $\mathbf{x}_i \in \mathcal{X}$ is an observation of \mathbf{X} . Then \mathcal{X} is the sample space of the random vector \mathbf{X} and the i th row of \mathbf{X} is the sample space of the i th random variable X_i (also called the i th random component) of \mathbf{X} .

For convenience, we assume that the random vector \mathbf{X} is normalized to have zero mean, that is, all of its components have zero mean: $E(X_i) = 0, 1 \leq i \leq D$. Recall that the variance of a random variable Z with n samples $\{z_i\}_{i=1}^n$ is computed by

$$\text{Var}(Z) = E((Z - \mu)^2) = \frac{1}{n} \sum_{i=1}^n (z_i - \mu)^2,$$

where μ is the mean of Z . Particularly, when $E(Z) = 0$,

$$\text{Var}(Z) = \frac{1}{n} \sum_{i=1}^n (z_i)^2 = \frac{1}{n} \|z\|^2.$$

In statistics, the first principal direction of the random vector \mathbf{X} (with zero mean) is defined as the direction of maximum variance. More precisely, we have the following.

Definition. Let \mathbf{X} be a D -dimensional random vector. Its first principal direction is the unit vector \mathbf{v}_1 defined by

$$\mathbf{v}_1 = \arg \max_{\|\mathbf{w}\|=1} \text{Var}(\mathbf{w}' \mathbf{X}). \quad (5.11)$$

Correspondingly, its first principal component is the random variable defined by

$$P_1 = \mathbf{v}_1' \mathbf{X}.$$

Similarly, assume that the $k - 1$ leading principal components of \mathbf{X} are Y_1, \dots, Y_{k-1} in the principal directions $\mathbf{v}_1, \dots, \mathbf{v}_{k-1}$, respectively. Then the k th principal direction is the unit vector determined by

$$\mathbf{v}_k = \arg \max_{\|\mathbf{w}\|=1} \text{Var} \left(\mathbf{w}' \left(\mathbf{X} - \sum_{i=1}^{k-1} \mathbf{Y}_i \mathbf{v}_i \right) \right)$$

and the k th principal component is the random variable

$$P_k = (\mathbf{v}_k)' \mathbf{X}.$$

If we make a k -dimensional random vector

$$\mathbf{P} = [P_1, \dots, P_k]',$$

which consists of k leading principal components of \mathbf{X} , then the sample space of \mathbf{P} is a subset of \mathbb{R}^k . Hence, from the statistical point of view, the PCA method is to find the d leading principal components of a given D -dimensional random vector \mathbf{X} . Assume that the rank of \mathbf{X} is r . Let the SVD of the data matrix \mathbf{X} be given by (5.6). Then the matrix \mathbf{U} in (5.6) gives the r -dimensional random vector $\mathbf{U} = [U_1, \dots, U_r]'$ such that each column is an observation of \mathbf{U} . We may re-write (5.6) in the random vector form:

$$\mathbf{X} = \sum_{i=1}^r (\sigma_i U_i) \mathbf{v}_i.$$

Recall that the singular values $\{\sigma_i\}_{i=1}^r$ are arranged in a descending order. By Theorem 5.2, the first principal component of \mathbf{X} is

$$P_1 = \sigma_1 U_1 \tag{5.12}$$

corresponding to the first principal direction \mathbf{v}_1 , which is the solution of (5.11). By (5.6), P_1 can also be written as

$$P_1 = (\mathbf{v}_1)' \mathbf{X}.$$

By the mathematical induction, we can prove that the leading d principal directions of \mathbf{X} are $\mathbf{v}_1, \dots, \mathbf{v}_d$, respectively, and the leading d principal components are

$$P_i = (\mathbf{v}_i)' \mathbf{X}, \quad 1 \leq i \leq d.$$

Representing \mathbf{P} in the sample space, we have

$$\mathbf{Y} = (\mathbf{V}_d)' \mathbf{X},$$

which is the same DR data \mathcal{Y} as in (5.10).

5.2 PCA Algorithms

Let $\mathbf{X} \in \mathfrak{M}_{D,n}$ be a given data matrix. We want to develop PCA algorithm to find its leading d principal directions and corresponding principal components, represented by a d -dimensional data matrix $\mathbf{Y} \in \mathfrak{M}_{d,n}$. By the discussion in the previous section, a PCA algorithm consists of two parts: data centralization and the SVD of the centered \mathbf{X} , where the SVD can be replaced by the spectral decomposition of its covariance matrix, particularly, when $D \ll n$. Besides the matrix \mathbf{Y} , the outputs of the algorithm also include (1) the d leading singular values of the centered \mathbf{X} , (2) the geometric center

of \mathcal{X} , and (3) the d principal directions $\{\mathbf{v}_1, \dots, \mathbf{v}_d\}$. The computation of the geometric center of \mathcal{X} is simple. When $D \ll n$, the spectral decomposition of the covariance matrix \mathbf{F} in (5.9) is applied and the DR data \mathbf{Y} is computed using the formula (5.10). When $D \geq n$, the (economic) SVD decomposition of the (centered) \mathbf{X} is applied to generating \mathbf{V}_d , $\boldsymbol{\Sigma}_d$ and \mathbf{U}_d , and then \mathbf{Y} is obtained using the formula (5.8).

5.2.1 Matlab Code for PCA Algorithm

If the dimension of the PCA kernel is not very large, for example, less than a couple of thousands, the MATLAB built-in eigen-decomposition functions work well for the spectral decomposition of a PCA kernel. We present the M-code of PCA DR algorithm in this situation as follows.

```

function [Y, v, Vd, mu] = drpca(X, d, options)
% Principal component analysis for DR.
%
% SYNTAX: [Y, v, Vd, mu] = DRPCA(X, d, options)
% INPUTS
%   X: (dim,n)-matrix whose columns are data points.
%   d: target dimension for DR data.
%   options: Options for eigen-computation.
% OUTPUTS
%   Y: (d, n)-matrix representing DR data.
%   v: (d, 1)-vector, d-leading eigenvalues of L.
%   Vd:(dim, d)-matrix, the d-leading eigenvectors of
%       the PCA kernel.
%   mu:(d, 1)-vector, the geometric center of X.
% Example: Reduce Swiss roll to 2-D.
% N=1000;
% tt = (3*pi/2)*(1+2*rand(1,N));
% height = 21*rand(1,N);
% X = [tt.*cos(tt); height; tt.*sin(tt)]; d=2;
% [Y,v,Vd,mu] = DRPCA(X, d, options);
%
% check input
[dim, n] = size(X);
if d > dim
error('d must be no more than data dimension.');
end
if ~isfield(options, 'disp')
options.disp = 0;
elseif ~isfield(options, 'isreal')
options.isreal = 1;
end
% centralizing data.
mu = mean(X,2);

```

```

X = X - repmat(mu,1,n);
% spectral decomposition
if dim <= n % case: dim<=n
% use covariance matrix.
L = cov(X',1);
if ~isfield(options, 'issym')
    options.issym = 1;
end
[Vd, ev] = eigs(L,d,'LM',options);
Y = sqrt(n)* Vd'*X;
v = diag(ev);
else % use SVD of X for dim>n
[Vd, S, U] = svds(X, d, options);
% compute Y first.
Y = S*U';
v = diag(S);
end

```

In the algorithm, the Matlab built-in function `eigs` (`svds` as well) adopts the Implicitly Restarted Arnoldi Method (IRAM), which was proposed by Lehoucq and Sorensen [4], to compute the leading eigenvalues and eigenvectors of a symmetric matrix. Before IRAM, many iterative eigensolvers had been developed—Jacobi method, Gauss-Seidel method, and Conjugate Gradient(CG) method, for example. Readers can find a comprehensive review of these methods in the SIAM book [5]. The Arnoldi iteration was invented by W. E. Arnoldi in 1951 [6] for finding the eigenvalues of general matrices. It is quite similar to the Lanczos iteration, which is mainly applied to solving the eigenvalue problems of Hermitian matrices. In the Arnoldi iteration method, eigenvalues are computed on a Krylov subspace by the QR algorithm. IRAM restarts the Arnoldi methods after some iteration to save the memory storage. IRAM algorithm can be obtained from the free software package ARPACK [4, 7, 8].

5.2.2 EM-PCA Algorithm

In 1998, S. Roweis developed an EM iterative algorithm for PCA [9]. In the EM-PCA algorithm, a few leading eigenvalues and corresponding eigenvectors are computed by a two-step iteration. The algorithm first finds the subspace spanned by the principal directions, then projects the observed data to the subspace, and finally makes a full singular value decomposition of the embedded data, obtaining the required principle components. The algorithm utilizes the EM iteration to find the principal subspace. Each iteration involves two steps: the expectation step (E-step) and the maximization step (M-step). In E-step, given the data matrix $\mathbf{X} \in \mathfrak{M}_{D,n}$ and the linear transformation $T \in \mathfrak{M}_{D,d}(d)$, the embedding data \mathbf{Y} so that $T\mathbf{Y} = \mathbf{X}$ is computed

by the least-square. In M-step, given the data matrix \mathbf{X} and the embedded data $\mathbf{Y} \in \mathfrak{M}_{d,n}(d)$, a best linear transformation T is computed by minimizing $\|T\mathbf{Y} - \mathbf{X}\|_F$. Alternately applying these two steps, we can reach the required subspace. The proof of the convergence of the EM iteration can be found in [10]. We now present a simple EM PCA M-code, whose original version can be found in <http://www.cs.nyu.edu/~roweis/code.html>.

```

function [Y, v, Vd, mu] = drempca(X, d, iter, T)
% EM principal component analysis.
%
% SYNTAX: [Y, v, Vd, mu] = DREMPCA(X, d, iter)
%
% INPUTS
%   X: (dim,n)-matrix whose columns are data points.
%   d: target dimension for DR data.
%   iter: number of EM iterations.
%   T: Initial of a mapping from R^dim(X) to R^d.
%
% OUTPUTS
%   Y: (d, n)-matrix representing DR data.
%   v: (d, 1)-vector, d-leading eigenvalues of L.
%   Vd:(dim, d)-matrix, the d-leading eigenvectors
%       of the PCA kernel.
%   mu:(d, 1)-vector, the geometric center of X.
%
% Example: Reduce Swiss roll to 2-D.
% N=1000;
% tt = (3*pi/2)*(1+2*rand(1,N));
% height = 21*rand(1,N);
% X = [tt.*cos(tt); height; tt.*sin(tt)]; d=2;
% [Y,v,Vd,mu] = DREMPCA(X, d, 20);

% Initialize inputs
[dim,n] = size(X);
if nargin<4
    T = randn(dim,d);
    if nargin<3
        iter = 20;
    end
end
assert(d<=dim);
if isempty(T)
    T = rand(dim,d);
else
    assert(d ==size(T,2));
    assert(dim ==size(T,1));
end
% Centralizing data
mu = mean(X,2);
X = X - repmat(mu,1,n);
% EM iteration

```

```

for i=1:iter
    % e step -- map X to R^d.
    %           (Solve TY=X for Y).
    Y = T\X;
    % m step -- find best data fitting.
    %           (Solve TY=X for T).
    T = X/Y;
end
% o.n. basis on subspace
T = orth(T);
% o.n. projection of X
tx = T'*X;
% covariance matrix of tx'
ctx = cov(tx',1);
% eigen-decomposition of cov(tx);
[vects,eigv] = eig(ctx);
S = fliplr(vects);
eigv = diag(eigv);
v = eigv(d:-1:1);
Vd= T*S;
Y = S'*tx;

```

The EM-PCA algorithm finds eigenvectors without explicitly decomposing the covariance matrix of data. It is very effective when d is small for its complexity is $O(Ddn)$ per iteration. As a comparison, we remark that a standard PCA method, if it explicitly computes the eigenvectors of the covariance matrix, has the complexity $O(D^2n)$, or, if it explicitly computes the eigenvectors of the Gram matrix, has the complexity $O(Dn^2)$. Hence, when $d \ll D$, the EM-PCA algorithm has the least complexity.

When the data is contaminated by noise, Roweis [9] modified EM-PCA to the Sensible Principal Component Analysis (SPCA) algorithm. Readers are referred to [9] for the details.

5.2.3 Testing PCA Algorithm on Artificial Surfaces

PCA is a linear projection method. When the data reside on a subspace or a hyperplane, it works well. However, if the data reside on a nonlinear manifold, it often fails. In this subsection, we demonstrate the validity of PCA on the data sets sampled from several artificial surfaces.

In Fig. 5.1, the graphs on the top are three surfaces, namely, the left is a hyperplane, the middle is a 3-D cluster, and the right is a punched sphere. The data of 3-D cluster consist of three separated spheres and two lines that link the spheres. Hence, 3-D cluster is not a smooth surface. From each surface 2,000 points are randomly selected to build the data set. Colors are used to show the neighborhood structure of the data. We use PCA to reduce

the data of these surfaces to 2-D data sets. Recall that a DR data set is valid if the neighborhood structures are preserved. Therefore, the colors should not be mixed on the DR data sets. In Fig. 5.1, three graphs at the bottom represent the DR data sets of the surfaces on the top. It is not surprise that the square data are well represented on plane since the original data are on a hyperplane. But the remaining two data sets are not well represented on plane, for mixtures of colors are found in the marked areas.

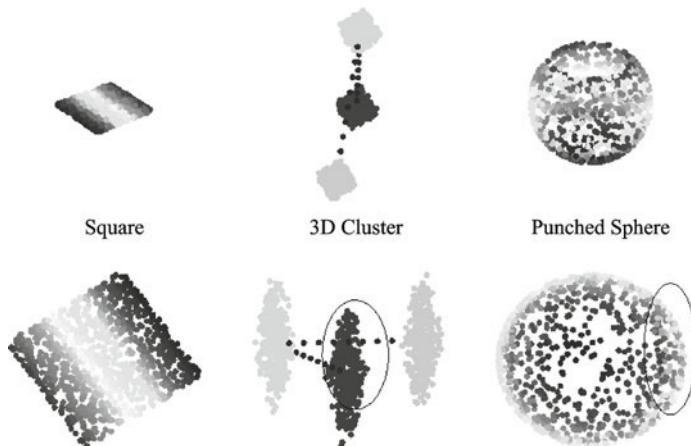


Fig. 5.1 PCA for three surfaces. On the top, the left is a square in the space, the middle is a 3-D cluster, and the right is a punched sphere. Their 2-D DR data sets are displayed at the bottom correspondingly. It is shown that PCA cannot make valid DR for 3-D cluster and punched sphere since these data are sampled from nonlinear surfaces.

In Fig. 5.2 and Fig. 5.3, we apply PCA to projecting two surfaces, Swiss roll and S-curve, onto the plane. Both of them are expandable surfaces. The lengths of the surfaces in Fig. 5.3 are longer than those in Fig. 5.2. Figure 5.3 shows that when the lengths of these surfaces are longer, the colors of the DR data are mixed in larger areas. Hence, PCA is not effective in the RD of these surfaces.

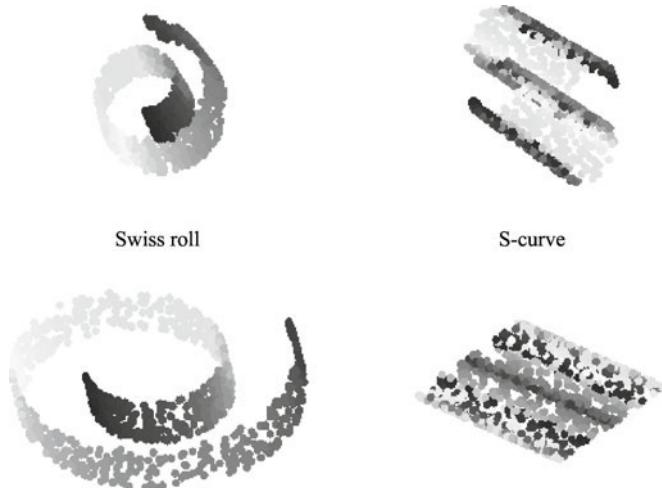


Fig. 5.2 PCA for Swiss roll and S-shape with shorter lengths. Swiss roll and S-shape are expandable surfaces. They can be isometrically unfolded to rectangles, but cannot be isometrically projected onto the plane. When their lengths are shorter, they can be projected onto the plane without significant distortion on the neighborhood structures of the data.

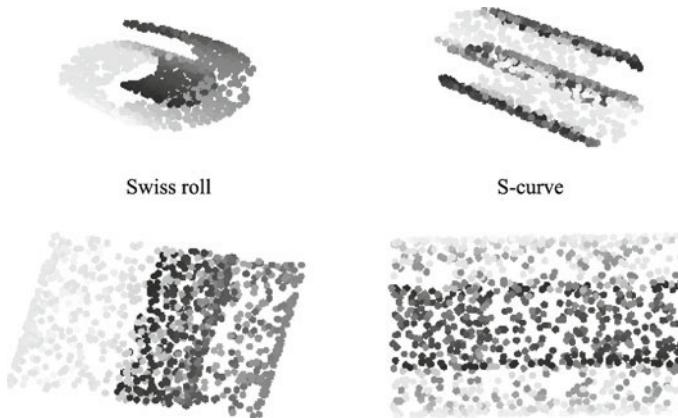


Fig. 5.3 PCA for Swiss roll and S-shape with longer lengths. When their lengths are long, we cannot find the projections that preserve the neighborhood structures of the data.

5.3 Applications of PCA

In Chapter 1, we have already briefly introduced the motivation of DR: It converts high-dimensional data to low-dimensional ones so that the low-

dimensional data can be effectively used in data processing systems. To understand how to apply DR technique in application, we give several examples.

5.3.1 PCA in Machine Learning

Let $\mathcal{X} \subset \mathbb{R}^D$ be a data set with the extrinsic dimension D . Assume that the size of the data set, denoted by $|\mathcal{X}|$, is very large. In machine learning, a main task is to find a (vector-valued) function f on \mathcal{X} , which captures characteristics of interest of the data. These characteristics can be used in certain applications, for example, in the classification of \mathcal{X} . We call such a function a *feature extractor* or a *feature function*, and call \mathcal{X} the *examples* of the learning and $f(\mathcal{X})$ the *features* of \mathcal{X} . A machine learning algorithm is applied to *learning* the feature extractor f from the examples \mathcal{X} . Assume that the characteristics of interest of a small subset of examples, say, $\mathcal{X}_1 \subset \mathcal{X}$ with $|\mathcal{X}_1| \ll |\mathcal{X}|$, are known in advance, i.e., $c = f(\mathbf{x})$ is known for each $\mathbf{x} \in \mathcal{X}_1$. We denote by \mathcal{C}_1 the set $f(\mathcal{X}_1) = \{f(\mathbf{x})\}_{\mathbf{x} \in \mathcal{X}_1}$. A learning algorithm then can be considered as an operator L , which *learns* the function f from the pair $(\mathcal{X}_1, \mathcal{C}_1)$: $f = L(\mathcal{X}_1, \mathcal{C}_1)$. This type of learning is called supervised learning. Write $\mathcal{X}_2 = \mathcal{X} \setminus \mathcal{X}_1$. Once f is learned, then, to extract the characteristics of interest of each vector $\mathbf{z} \in \mathcal{X}_2$, we simply evaluate f at \mathbf{z} to get its features $f(\mathbf{z})$. Hence, the whole machine learning processing has two steps: (1) constructing the feature extractor f via the examples and (2) evaluating f at unknown data. Therefore, a (supervised) machine learning algorithm can be summarized by the diagrams in Fig. 5.4. If the dimension of data is high, these two steps in the processing are involved in high-dimensional computation. To avoid the high-dimensional data in the processing, DR technique can be applied. On the example set \mathcal{X}_1 , we first determine the intrinsic dimension d of \mathcal{X}_1 , then construct a DR mapping h , which maps \mathcal{X}_1 onto $\mathcal{Y}_1 \subset \mathbb{R}^d$: $h(\mathbf{x}) = \mathbf{y}$. Since \mathcal{Y}_1 preserves the characteristics of \mathcal{X}_1 , we can *learn* the feature extractor from \mathcal{Y}_1 . Let the new feature extractor learned from \mathcal{Y}_1 be denoted by g . Ideally, we shall have $f(\mathbf{x}) = g(\mathbf{z})$. Extend the DR mapping h on the example set \mathcal{X}_2 and write $\mathcal{Y}_2 = h(\mathcal{X}_2)$. Then extend the extractor g on \mathcal{Y}_2 . We now can learn the features of \mathcal{X}_2 via the DR set \mathcal{Y}_2 . Hence, DR technique enables us to convert the learning process on the high dimensional examples in Fig. 5.4 to the process on the DR data sets \mathcal{Y}_1 and \mathcal{Y}_2 respectively. Since the new data processing performs on low-dimensional sets, it enables us to overcome the difficulties caused by the curse of dimensionality.

Note that a DR algorithm itself serves as an unsupervised learning algorithm. In an unsupervised learning problem, the characteristics of interest of the example set \mathcal{X}_1 are unknown and must be learned from the examples. The projection in PCA can also be considered as a feature extractor, which extracts the principal components of the data. The principal components

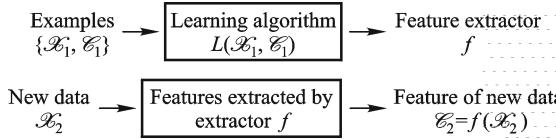


Fig. 5.4 The data feature extraction via machine learning. Top: A learning algorithm is used to find a feature extractor from examples. Bottom: The learned feature extractor is applied to extracting the features of the new data.

present features of the data.

We now introduce some applications, where PCA plays the role of either a feature extractor or a data simplifier. Due to the linearity of PCA, using PCA to reduce data dimension and extend feature functions is computationally economic. Hence, PCA is widely applied in real-world applications which need the feature extractions.

5.3.2 PCA in Eigenfaces

Eigenface [11, 12, 13] is an important method in face recognition. The task of face recognition is discriminating input face images into people in a set. Developing a computational model of face recognition is quite difficult, because faces are complex, multidimensional, and meaningful visual stimuli. Surely, the input images have observed patterns, which can be the presence of some objects such as eyes, nose, mouth, as well as relative distances between these objects. However, the approaches focusing on detecting individual features have been proved difficult to extend to multiple views, and have often been quite fragile. The eigenface approach is an information theory approach of coding and decoding face images that may give insight into the information content of face images. That is, we want to encode the relevant information in a face image as effectively as possible, and compare one face encoding with a database of models encoded similarly. Assume that we have a database containing a vast amount of face images of a group of people, of which each person has a lot of various faces. In machine learning, the database is often called a *training set*. Therefore, the first step in face recognition is to find the principal components of the distributed faces in the training set. Mathematically these components are the eigenvectors of the covariance matrix of the face images, where each image is treated as a random variable. These eigenvectors are called eigenfaces. Since eigenfaces are principal components of the set of face images, a face in the set, after subtracting the average face, can be represented as a weight sum of the eigenfaces within a tolerant energy loss. The space spanned by eigenfaces is called *face space*. Each image in the database now is a point in the face space. In application, the dimension of the face space is quite low, usually only 40-60. After the average face and

the eigenfaces are computed, each input image can be represented as the sum of average face, weighted eigenfaces, and remainder. The remainder measures the distance from the input image to the face space, which is used to determine whether the input image is a face. Once the input image is recognized as a face, the weights on eigenfaces are used to determine whether the input image is a known face (in the database) or a new face.

For demonstration, we present a simplest eigenface algorithm. The algorithm consists of three steps, using the training set as the input. Let the training set be denoted by \mathbf{X} . As explained in PCA, each row vector of \mathbf{X} is a random variable representing a face image.

Step 1. Construction of eigenfaces. Let the random vector $\mathbf{X} = [X_1; \dots; X_N]$ represent the training set, in which each random variable X_j represents a face image. We identify the notation $\mathbf{X} = [X_1; \dots; X_N]$ with the matrix \mathbf{X} whose rows are X_j , $1 \leq j \leq n$. Applying PCA on \mathbf{X} , we obtain the average face $\bar{\mathbf{X}}$, the eigenfaces F_1, \dots, F_d , and the $d \times N$ weight matrix $\mathbf{Y} = [\mathbf{y}_1 \ \dots \ \mathbf{y}_N]$. The eigenfaces form an o.n. basis of the face space S_d . In the weight matrix, the i th column $\mathbf{y}_i = [y_{1i}, \dots, y_{di}]'$ is the weight vector, which approximatively represents the i th face X_i in the training set such that $X_i \approx \sum_{j=1}^d y_{ji} F_j$.

Step 2. Computation of the weights. Compute the weights of a new input face with respect to eigenfaces and the distance from the new input face to the face space. Assume that \mathbf{Z} is an input image (represented by a row vector). The weights of \mathbf{Z} with respect to eigenfaces are computed by

$$\mathbf{z} = \mathbf{F}' * (\mathbf{Z} - \bar{\mathbf{X}}),$$

and the distance from \mathbf{Z} to the face space S_d is computed by

$$d_1 = \|\mathbf{Z} - (\mathbf{z}' \mathbf{F} + \bar{\mathbf{X}})\|.$$

Step 3. Face recognition. Let ε_1 and ε_2 be the thresholds for determining whether an input image is a face and whether it is a known face, respectively. If $d_1 < \varepsilon_1$, then Z is a face. Otherwise, it is not. In the case that Z is a face, compute the sum of distances between the weight vector \mathbf{z} and the weight vectors of examples by

$$d_2 = \sum_{1 \leq i \leq n} (\mathbf{z} - \mathbf{y}_i)^2.$$

If $d_2 < \varepsilon_2$, Z is assumed to be the face of a person in the training set, and the person is recognized from the nearest neighbors of Z . Otherwise, it is assumed to be a new face.

Remark 5.2. In the algorithm, the distances d_1 and d_2 can be computed by using other distances other than the Euclidean one.

The face images (see Fig. 5.5) used in the demonstration are chosen from the UMIST Face Database in the web-site tutorial-haartraining.googlecode.com/

svn/trunk/data/umist_cropped. We demonstrate eigenfaces obtained by PCA in Fig. 5.6. The recovered results are shown in Fig. 5.7.



Fig. 5.5 The face images in a training set are chosen from the UMIST Face Database-1a. The faces are ordered from left to right, and top to bottom.



Fig. 5.6 The average face and eigenfaces for the images in Fig. 5.5 are presented. The left most one is the average face, and the others are eigenfaces.

Eigenface technique provides a simple approach to face recognition. Based on other feature-driven representations of faces, many algorithms have been developed. Researchers in the area of face recognition believe that the features of face images can be roughly divided into two categories: small-scale features and large-scale features. The small-scale features are conceived as being illumination invariant, while the large-scale features are sensitive to illumination. Methods based on extracting small-features include logarithmic



Fig. 5.7 The faces in the training set in Fig. 5.5 are recovered by the average face and eigenfaces in Fig. 5.6.

mic total variation model [14], self quotient image method [15], and discrete wavelet transform method [13]. It has been observed that the effect of illumination variations in face images sometimes is more significant than the effect of the image variations due to the change in face identity. Therefore, the normalization of face illumination becomes an important problem in face recognition and face image processing. Since illumination mainly impacts the large-scale features of face images, methods based on extracting large-scale features have to compensate illuminated images. Representative algorithms based on extracting large features include histogram equalization [16], shape-from-shading [17], illumination cone [12], linear illumination model and low-dimensional illumination space representation [18, 19], and the non-point light quotient image relighting method based on PCA [20, 21]. However, a feature-based algorithm often meets the problem of inconsistency of small-scale features and large-scale features. Usually the methods based on preserving large-scale features will distort the small-scale features and vice versa. Xie, Zheng, Lai, and Yuen [22] proposed a new framework that combines large-scale and small-scale features together to generate normalized

face images. For details in this aspect, refer to [22–25] and their references.

5.3.3 PCA in Hyperspectral Image Analysis

Hyperspectral sensors collect information as a set of images in different bands. A hyperspectral image cube can have images over 200 bands. Each pixel (also called a raster cell) in the cube contains a vector called a *hyperspectral signature*, or a *spectral curve*, which represents a certain material. Hyperspectral images (HSI) can be used for geology, forestry and agriculture mapping, land cover analysis, atmospheric analysis, and other applications. Directly using hyperspectral data in data analysis processing is too expensive because of the huge data size. It is noted that the correlations between different bands of HSI are very high. Hence, it is possible to map hyperspectral images of all bands into a small cube of hyperspectral *eigenimages* (less bands) while the main features of the original data are still preserved. Principal component analysis is an effective method to reduce the number of bands of hyperspectral image data.

Let us denote an HSI cube by a matrix array:

$$\mathbf{H} = \{h_{j,k,l}; \quad 1 \leq j \leq J, 1 \leq k \leq K, 1 \leq l \leq D\},$$

where for each fixed l , the data represent the l th-band (spatial) image, while for each fixed order-pair (j, k) , the data represent the spectral curve of the raster cell at the position (j, k) . In order to apply PCA algorithm to the HSI data, we first convert the HSI cube to a matrix using a one-to-one index transformation $\mathbb{N}^2 \rightarrow \mathbb{N}$, which uniquely maps each double-index pair (j, k) to a single index $i \in \{1, 2, \dots, n\}$, with $n = J \times K$. Let the converted $D \times n$ matrix be denoted by $\mathbf{P} = (p_{l,i})_{D \times n}$, where $p_{l,i} = h_{j,k,l}$ and i is associated with (j, k) according to the index transformation. Each column of \mathbf{P} represents the spectral curve of the corresponding raster cell while each row of \mathbf{P} is a representation of a band image. For convenience, we shall denote the l th row of \mathbf{P} by \mathbf{P}_l . Let the average (hyperspectral) image be denoted by $\bar{\mathbf{P}}$ and set $\tilde{\mathbf{P}}_l = \mathbf{P}_l - \bar{\mathbf{P}}$. Then the covariance matrix with respect to band images is the $D \times D$ matrix

$$\mathbf{C} = \frac{1}{n} \tilde{\mathbf{P}} \tilde{\mathbf{P}}^T.$$

Using PCA, we obtain d principal components, say, B_1, \dots, B_d , which are called *virtual band-images*, and a $d \times D$ weight matrix $\mathbf{W} = [w_{i,j}]_{i,j=1}^{d,D}$ such that each band-image in the original HSI cube can be approximately represented by

$$\mathbf{P}_l = \bar{\mathbf{P}} + \sum_{i=1}^d w_{i,l} \mathbf{B}_i. \quad (5.13)$$

The HSI data processing now can be applied to the virtual band-images so that it will perform faster and more effectively.

PCA can also be applied in other applications, such as document ordering and fingerprint classification. Since the approaches are similar, we skip them here.

References

- [1] Pearson, K.: On lines and planes of closest fit to systems of points in space. *Philosophical Magazine* 2(6), 559–572 (1901).
- [2] Jolliffe, I.T.: Principal Component Analysis. Springer Series in Statistics. Springer-Verlag, Berlin (1986).
- [3] Rao, C., Rao, M.: Matrix Algebra and Its Applications to Statistics and Econometric. World Scientific, Singapore (1998).
- [4] Lehoucq, R., Sorensen, D.: Deflation techniques for an implicitly re-started arnoldi iteration. *SIAM J. Matrix Analysis and Applications* 17, 789–821 (1996).
- [5] Barrett, R., Berry, M.W., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., van der Vorst, H.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM Publisher (1993).
- [6] Arnoldi, W.E.: The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics* 9, 17–29 (1951).
- [7] Lehoucq, R., Sorensen, D., Yang, C.: ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods. SIAM Publications, Philadelphia (1998).
- [8] Sorensen, D.: Implicit application of polynomial filters in a k-step arnoldi method. *SIAM J. Matrix Analysis and Applications* 13, 357–385 (1992).
- [9] Roweis, S.: EM algorithms for PCA and SPCA. In: *anips*, vol. 10, pp. 626–632 (1998).
- [10] Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, B* 39(1), 1–38 (1977).
- [11] Belhumeur, P.N., Hespanha, J.P., Kriegman, D.J.: Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *TPAMI* 7(19), 711–720 (1997).
- [12] Georghiades, A., Belhumeur, P., Kriegman, D.: From few to many: Illumination cone models for face recognition under variable lighting and pose. *TPAMI* 6(23), 643–660 (2001).
- [13] Wang, W., Song, J., Yang, Z., Chi, Z.: Wavelet-based illumination compensation for face recognition using eigenface method. In: Proceedings of Intelligent Control and Automation. Dalian, China (2006).
- [14] Chen, T., Zhou, X.S., Comanicu, D., Huang, T.: Total variation models for variable lighting face recognition. *TPAMI* 9(28), 1519–1524 (2006).
- [15] Wang, H.T., Li, S.Z., Wang, Y.S.: Face recognition under varying lighting conditions using self quotient image. In: Automatic Face and Gesture Recognition, International Conference on FGR, pp. 819–824 (2004).
- [16] Basri, R., Jacobs, D.: Photometric stereo with general, unknown lighting. In: IEEE Conference on CVPR, 374–381 (2001).

- [17] Horn, B.: The Psychology of Computer Vision, chap. 4. Obtaining Shape from Shading Information and chap 6. Shape from Shading, pp. 115–155. McGraw-Hill, New York (1975).
- [18] Xie, X.D., Lam, K.: Face recognition under varying illumination based on a 2D face shape model. *Journal of Pattern Recognition* 2(38), 221–230 (2005).
- [19] Hu, Y.K., Wang, Z.: A low-dimensional illumination space representation of human faces for arbitrary lighting conditions. In: *Proceedings of ICPR*, pp. 1147–1150. Hong Kong (2006).
- [20] Ramamoorthi, R.: Analytic PCA construction for theoretical analysis of lighting variability in images of a lambertian object. *TPAMI* 10(24), 1322–1333 (2002).
- [21] Wang, H.T., Li, S.Z., Wang, Y.: Generalized quotient image. In: *Proceeding of CVPR* (2004).
- [22] Xie, X., Zheng, W.S., Lai, J., Yuen, P.C.: Face illumination normalization on large and small scale features. In: *IEEE Conference on CVPR* (2008).
- [23] Xie, X., Lai, J., Zheng, W.S.: Extraction of illumination invariant facial features from a single image using nonsubsampled contourlet transform. *Pattern Recognition* 43(12), 4177–4189 (2010).
- [24] Xie, X., Zheng, W.S., Lai, J., Yuen, P.C., Suen, C.Y.: Normalization of face illumination based on large- and small- scale features. *IEEE Trans. on Image Processing* (2010). Accepted.
- [25] Zheng, W.S., Lai, J., Yuen, P.C.: Penalized pre-image learning in kernel principal component analysis. *IEEE Trans. on Neural Networks* 21(4), 551–570 (2010).
- [26] Turk, M., Pentland, A.: Eigenfaces for recognition. *The Journal of Cognitive Neuroscience* 3(1), 71–86 (1991).

Chapter 6 Classical Multidimensional Scaling

Abstract Classical multidimensional scaling (CMDS) is a technique that displays the structure of distance-like data as a geometrical picture. It is a member of the family of MDS methods. The input for an MDS algorithm usually is not an object data set, but the similarities of a set of objects that may not be digitalized. The input distance matrix of CMDS is of Euclidean-type. There exists an r -dimensional vector set \mathcal{X} such that the Euclidean distance matrix of \mathcal{X} is equal to the input one. The set \mathcal{X} is called a configuration of the input matrix. In the case that the dimension of the set \mathcal{X} is too high to be visualized, we then need to reduce the dimension of the configuration to 2 or 3 for visualization. Sometimes, a little higher dimension is acceptable. CMDS is equivalent to PCA when the input of CMDS is a data set. The data model in CMDS is linked to a complete weighed graph, in which the nodes are objects and the weights are the similarities between the objects. This chapter is organized as follows. In Section 6.1, we introduce multidimensional scaling. In Section 6.2, we discuss the relation between Euclidean distance matrices and Gram matrices. The description of CMDS is in Section 6.3. The algorithm of CMDS is presented in Section 6.4.

6.1 Introduction of Multidimensional Scaling

6.1.1 Data Similarities and Configuration

A brief summary of the major types of multidimensional scaling (MDS), the distance models used by MDS, the similarity data analyzed by MDS, and the computer programs that implement MDS can be found in [1], where several applications are also introduced. In this section, we introduce the data models used in MDS and the concepts of MDS.

MDS is a set of data analysis techniques that display the structure of

distance-like data as a geometrical picture. It is a powerful tool in data visualization and other data processing areas. MDS has its origins in psychometrics, where it was proposed to help to understand people's judgments of the similarity of members of a set. Torgerson in [2] proposed the first MDS method and coined the term. MDS has now become a general data analysis technique used in a wide variety of fields, such as marketing, sociology, physics, political science, biology, and others [3].

MDS pictures the structure of a set of objects from data that approximate the distances between pairs of the objects. The data are called similarities or dissimilarities, that reflect the amount of similarity or dissimilarity between pairs of the objects.

In addition to the traditional human similarity judgment, the data can also be an "objective" similarity measure, depending on the similarity of interest. In mathematics and statistics, we often use *distance* to describe the dissimilarity between two objects, and use *covariance* to describe their similarity. Perhaps, distance measurement is more commonly used in application because it is easy to measure.

In MDS, the term *distance* is used in a wider sense, standing for *measure of dissimilarity*. Distance/proximity must reflect the amount of dissimilarity/similarity between pairs of the objects. We shall use the term *similarity* generically to refer to both *similarities* (where large numbers refer to great similarity) and *dissimilarities* (where large numbers refer to great dissimilarity).

To better understand the similarities used in MDS, we explain the concept in more detail. Assume that there are n objects in a set and the similarities between all pairs are measured. Then we represent the measured similarities by an $n \times n$ dense matrix \mathbf{D} , called the *similarity matrix*. In some applications, not all pairs can be measured. Then, we only obtain a sparse similarity matrix, where a non-diagonal 0-entry indicates that the corresponding similarity is not measured. Sometimes, an index is calculated from multivariate data, then the similarities between objects need to be computed for each component of the index. For example, the proportion of agreement in the votes cast by several senators is an index. Therefore, there are more than one similarity matrices created. We may weight each similarity matrix and then combine the matrices into a single distance matrix. In some cases, we need to compute distance matrix for each component so that several distance matrices are created. Similar situation happens when we want to picture the "flight relations" among cities. The flight time can be used to measure similarities between cities. We can use the flight cost or flight distance to measure the similarities too. Therefore, the similarities between cities are represented by a matrix array. Thus, we may use the array to compute a single distance matrix to represent the similarities between cities, or directly use the matrix array to describe the similarities.

MDS pictures the structure of a set of objects using configuration point set in a Euclidean space. As we introduced in Chapter 4, if objects can be

digitized to a vector in a Euclidean space, then we obtain the data of the first type. In some cases, the objects are hard to be digitized. Most of objects in sociology, political science, and biology are in this category. Then the objects are configured as virtual points in a Euclidean space for their visualization or further processing. This point set is called *configuration*. The configurative points are arranged so that the Euclidean distances between them have the closest relation to the similarities between the objects. That is, two similar objects are represented by two virtual points that are close to each other, and two dissimilar objects are represented by two virtual points that are far apart from each other. In visualization, the configurative points are 2-dimensional or 3-dimensional.

In summary, in an MDS processing, the input is a similarity matrix or an array of similarity matrices, and the output is a low-dimensional data set called *configuration*, which usually is 2- or 3-dimensional. The Euclidean distances between the configurative points must reflect the similarities of the objects, which are represented by the input.

6.1.2 Classification of MDS

MDS is a generic term that includes many different specific types. These types can be classified according to whether the similarities data are qualitative (called nonmetric MDS) or quantitative (called metric MDS). The number of similarity matrices and the nature of the MDS models can also be used to classify MDS, yielding classical MDS (one similarity matrix in an unweighted data model), replicated MDS (several similarity matrices in an unweighted model), and weighted MDS (several similarity matrices in a weighted model).

Metric MDS

The term *Metric MDS*, shortly, MMDS, has been used in slightly different meanings. In some references, it is used in a narrow sense, identified with *Classical MDS* (CMDS). The identifying aspect of CMDS is that there is only one similarity matrix and Euclidean distance is applied to computing the similarities. It is also assumed that in the distance matrix there are no missing entries, i.e., the distances between all pairs are measured. The input data for classical MDS may come from two different resources. If the objects are digitized to D -dimensional vectors that form a data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$, then we can compute the Euclidean metric $\mathbf{D} = [d_{ij}]_{i,j=1}^n$ in the standard way, using the formula

$$d_{ij} = d_2(\mathbf{x}_i, \mathbf{x}_j).$$

The matrix \mathbf{D} is also called the *similarities* in classical MDS. In the case that the set of objects is not presented, but the similarities \mathbf{D} , known as a Euclidean metric, is available, then \mathbf{D} is the input of CMDS. Classical MDS can be performed efficiently on large matrices. However, if the similarity

matrix \mathbf{D} is not a Euclidean metric, no simple solution can be obtained as in CMDS. In this case, \mathbf{D} may represent other metrics. In a general sense, Metric MDS (MMDS) is the technique for finding configuration using general metrics. For instance, the Minkowski metrics (also called l^p distance),

$$d_{ij} = d_p(\mathbf{x}_i, \mathbf{x}_j) \stackrel{\text{def}}{=} \left(\sum_{k=1}^D |x_{ki} - x_{kj}|^p \right)^{1/p},$$

are often applied. The Manhattan distance, also called city block distance, is also often used in a Metric MDS processing.

Nonmetric MDS

Nonmetric MDS is first devised by Shepard [4, 5]. The goal of nonmetric MDS is to establish a monotonic relationship between interpoint distances and obtained similarities. The advantage of nonmetric MDS is that no assumptions need to be made about the underlying transformation function. The only assumption is that the data is measured at the ordinal level.

MDS can also be classified by number of similarity matrices.

Classical MDS

As introduced in Metric MDS, classical MDS uses a single distance matrix. Usually the Euclidean metric is applied.

Replicated MDS

Several similarity matrices are input in replicated MDS (RMDS) and then they are analyzed simultaneously. In RMDS, these similarity matrices are used to produce a single distance matrix, which is further applied in RMDS procedure. The output of RMDS may be several configurative point sets, representing different groups of interest.

Weighted MDS

The input of weighted MDS (WMDS) is quite similar to RMDS. Several similarity matrices are input in weighted MDS and they are analyzed simultaneously. Unlike RMDS, in WMDS, these similarity matrices are not merged into a single distance matrix, but used to produce several weighted distance matrices for WMDS procedure. The output of WMDS also consists of several configurative point sets, representing different groups of interest. But each of them is derived from its own distance matrix.

6.2 Euclidean Matric and Gram Matrices

In the classical MDS, the similarities between the original objects are assumed to be represented by the Euclidean metric in the form of a Euclidean distance

matrix. The task of CMDS is to find a configuration $\mathcal{Y} \subset \mathbb{R}^d$ such that the Euclidean distance matrix of \mathcal{Y} best approximates the given similarities.

From the viewpoint of geometry, in a Euclidean space, the distance describes the dissimilarity of a pair of points while the inner product describes the similarity. They have a close relationship. In this section we study the relation between Euclidean metric and the Gram matrix.

In the following, we represent a data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ in the matrix form $\mathbf{X} = [\mathbf{x}_1 \ \cdots \ \mathbf{x}_n] \in \mathfrak{M}_{D,n}$, where each column of \mathbf{X} is a point of \mathcal{X} . Thus, the totality of all columns of the data matrix \mathbf{X} constitutes the data set \mathcal{X} . For convenience, we shall identify the data matrix \mathbf{X} with the data set \mathcal{X} .

6.2.1 Euclidean Distance Matrices

Recall that the Euclidean distance between two vectors $\mathbf{a} = [a_1, a_2, \dots, a_D]'$ and $\mathbf{x} = [x_1, x_2, \dots, x_D]'$ in the space \mathbb{R}^D is defined by

$$d_2(\mathbf{a}, \mathbf{x}) = \|\mathbf{x} - \mathbf{a}\| = \sqrt{\sum_{l=1}^D (x_l - a_l)^2}.$$

The Euclidean inter-point distance matrix, or simply the Euclidean distance matrix, on the data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ is

$$\mathbf{D} \stackrel{\text{def}}{=} [D_{ij}] = [d_2(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^n, \quad (6.1)$$

which defines a metric on \mathcal{X} called Euclidean metric. We also need the Euclidean square-distance matrix (see Section 3.1)

$$\mathbf{S} = [d_2^2(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^n.$$

Both \mathbf{D} and \mathbf{S} are symmetric, and also invariant of shift and rotation. Indeed, assume that $\mathbf{a} \in \mathbb{R}^D$, and $\mathbf{R} \in \mathfrak{O}_D$ is an o.g. matrix. Recall that the \mathbf{a} -shift of \mathcal{X} is the set $\mathcal{X}_{\mathbf{a}} = \{\mathbf{x} + \mathbf{a} : \mathbf{x} \in \mathcal{X}\}$, and the R -rotation of \mathcal{X} is the set $\mathcal{X}_R = \{R\mathbf{x} : \mathbf{x} \in \mathcal{X}\}$. It is easy to see that the Euclidean distance matrices of $\mathcal{X}_{\mathbf{a}}$ and \mathcal{X}_R are the same as those of \mathbf{X} . We now define the Euclidean metric in a general sense.

Definition 6.1. An $n \times n$ symmetric matrix \mathbf{D} is called a Euclidean distance matrix (or Euclidean metric) if there exists an integer $m > 0$ and a vector set $\mathcal{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_n\} \subset \mathbb{R}^m$ so that

$$\mathbf{D} \stackrel{\text{def}}{=} [D_{ij}] = [d_2(\mathbf{z}_i, \mathbf{z}_j)]_{i,j=1}^n. \quad (6.2)$$

The vector set \mathcal{Z} is called a configurative point set (or a configuration) of \mathbf{D} .

We often need to determine whether a matrix is a Euclidean metric directly by the properties of the matrix, without the aid of its configuration. For this purpose, we establish the relation between Euclidean metric and psd matrix.

6.2.2 Gram Matrix on Data Set

Recall that in the Euclidean space \mathbb{R}^D the inner product of two vectors \mathbf{x} and \mathbf{y} is defined by

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^D x_i y_i,$$

and the Gram matrix on the data set \mathcal{X} is defined by

$$\mathbf{G} = [G_{ij}] = [\langle \mathbf{x}_i, \mathbf{x}_j \rangle]_{i,j=1}^n.$$

It is clear that G is a positive semi-definite (psd) matrix. On the other hand, each psd matrix represents a Gram matrix of a certain data set. Indeed, if an $n \times n$ psd matrix G has rank m , then it has a Cholesky decomposition

$$\mathbf{G} = \mathbf{X}' \mathbf{X}, \quad (6.3)$$

where $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_n]$ is an $m \times n$ matrix. By (6.3), G is the Gram matrix of the data set $\mathcal{X} \subset \mathbb{R}^m$. Therefore, we can identify a Gram matrix with a psd matrix.

6.2.3 Relation between Euclidean Distance and Gram Matrix

We now reveal the relation between the Gram matrix \mathbf{G} and the Euclidean distance matrix \mathbf{D} of a data set \mathcal{X} . Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^D$. By the Law of Cosines,

$$d_2(\mathbf{x}, \mathbf{y}) = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle - 2 \langle \mathbf{x}, \mathbf{y} \rangle},$$

which yields

$$D_{ij} = d_2(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{G_{ii} + G_{jj} - 2G_{ij}}. \quad (6.4)$$

The entries of a Gram matrix \mathbf{G} are vector inner products which are not shift invariant. In order to establish a relation between \mathbf{G} and \mathbf{D} , we shift the data \mathcal{X} by its center. Assume that \mathcal{X} resides on a d -dimensional hyperplane $H \subset \mathbb{R}^D$. Then the center of \mathcal{X} : $\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$ is in H , and $S = H - \bar{\mathbf{x}} \subset \mathbb{R}^D$ is a d -dimensional subspace parallel to H . Let the $\bar{\mathbf{x}}$ -shift of \mathcal{X} be denoted by

$$\hat{\mathcal{X}} = \{\hat{\mathbf{x}}_1, \dots, \hat{\mathbf{x}}_n\},$$

where $\hat{\mathbf{x}}_i = \mathbf{x}_i - \bar{\mathbf{x}}$. We call $\hat{\mathcal{X}}$ a *centered data set* and call the corresponding data matrix $\hat{\mathbf{X}}$ the *centered data matrix*.

Definition 6.2. For a given data set \mathcal{X} , let $\hat{\mathcal{X}}$ be the centered data set of \mathcal{X} . Then the Gram matrix of $\hat{\mathcal{X}}$:

$$\mathbf{G}^c = [\langle \hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j \rangle]_{i,j=1}^n = \hat{\mathbf{X}}' \hat{\mathbf{X}}, \quad (6.5)$$

is called the centering Gram matrix of \mathcal{X} .

In general, given $\mathbf{a} \in \mathbb{R}^D$, the \mathbf{a} -shifted Gram matrix of \mathcal{X} is defined by

$$\mathbf{G}^a = [\langle \mathbf{x}_i - \mathbf{a}, \mathbf{x}_j - \mathbf{a} \rangle]_{i,j=1}^n = \mathbf{X}'_a \mathbf{X}_a.$$

It is easy to verify that the centering Gram matrix \mathbf{G}^c and the Gram matrix \mathbf{G} have the same relationship with the Euclidean distance matrix \mathbf{D} described in (6.4), namely,

$$D_{ij} = \sqrt{G_{ii}^c + G_{jj}^c - 2G_{ij}^c}. \quad (6.6)$$

Since the centering Gram matrix plays an important role in data representation, we further elaborate our discussion on this topic.

Definition 6.3. Write $\mathbf{1} = [1, 1, \dots, 1]' \in \mathbb{R}^n$, $\mathbf{E} = \mathbf{1}\mathbf{1}'$, and let \mathbf{I} denote the $n \times n$ identity matrix. Then the $n \times n$ matrix $\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{E}$ is called the n -centralizing matrix.

If the dimension n is understood, we shall simplify the term “ n -centralizing” to “centralizing”.

Lemma 6.1. *The centralizing matrix \mathbf{H} has the following properties.*

- (1) $\mathbf{H}^2 = \mathbf{H}$,
- (2) $\mathbf{1}'\mathbf{H} = \mathbf{H}\mathbf{1} = \mathbf{0}$,
- (3) \mathcal{X} is a centered data set, if and only if $\mathbf{X}\mathbf{H} = \mathbf{X}$,
- (4) A psd matrix \mathbf{C} is a centering Gram matrix, if and only if $\mathbf{H}\mathbf{C}\mathbf{H} = \mathbf{C}$.

Proof. Since $\mathbf{1}'\mathbf{1} = n$, we have $\mathbf{E}\mathbf{1} = n\mathbf{1}$, so that

$$\mathbf{H}^2 = \left(\mathbf{I} - \frac{1}{n}\mathbf{E} \right)^2 = \mathbf{I} - \frac{2}{n}\mathbf{E} + \frac{1}{n^2}\mathbf{E}\mathbf{1}\mathbf{1}' = \mathbf{I} - \frac{1}{n}\mathbf{E} = \mathbf{H},$$

which yields (1). Furthermore, (2) follows from the fact that $\mathbf{E}\mathbf{1} = n\mathbf{1}$, (3) can be derived from the definition of centered data, and (4) is a direct consequence of (3). \square

By applying these properties directly, we have the following.

Lemma 6.2. *Let \mathbf{X} be a data matrix and \mathbf{G} be its Gram matrix. Then the centered data set of \mathbf{X} is $\mathbf{X}\mathbf{H}$, and the centering Gram matrix of \mathbf{X} is $\mathbf{G}^c = \mathbf{H}\mathbf{G}\mathbf{H}$.*

In general, the *centering matrix* of a symmetric matrix \mathbf{A} (not necessary psd) is defined as $\mathbf{A}^c = \mathbf{H}\mathbf{A}\mathbf{H}$. It is obvious that a symmetric matrix \mathbf{S} is

centering if and only if $\mathbf{S} = \mathbf{HSH}$. The notion of centering symmetric matrix enables us to represent the centering Gram matrix in terms of Euclidean square-distance matrix, reducing the relation (6.4) to a very simple form.

Theorem 6.1. *Euclidean square-distance matrix \mathbf{S} and the centering Gram matrix \mathbf{G}^c of a data set \mathcal{X} have the following relation.*

$$\mathbf{G}^c = -\frac{1}{2}\mathbf{S}^c. \quad (6.7)$$

Proof. It follows from Lemma 6.1 that \mathbf{G}^c has the property $\sum_{i=1}^n G_{ij}^c = 0$. Hence, the relation in (6.6) immediately yields both

$$\sum_{i=1}^n D_{ij}^2 = nG_{jj}^c + \sum_{i=1}^n G_{ii}^c$$

and

$$\sum_{j=1}^n D_{ij}^2 = nG_{ii}^c + \sum_{j=1}^n G_{jj}^c.$$

Therefore, the (i, j) -entry of \mathbf{S}^c is given by

$$\begin{aligned} (\mathbf{S}^c)_{ij} &= D_{ij}^2 - \frac{1}{n} \left(\sum_{i=1}^n D_{ij}^2 + \sum_{j=1}^n D_{ij}^2 - \frac{1}{n} \sum_{i,j=1}^n D_{ij}^2 \right) \\ &= D_{ij}^2 - G_{ii}^c - G_{jj}^c \\ &= -2G_{ij}^c, \end{aligned}$$

completing the proof of the theorem. \square

The following is a consequence of Theorem 6.1 and (6.3).

Theorem 6.2. *Let \mathbf{A} be a symmetric matrix. Then*

- (1) \mathbf{A} is a Gram matrix of a data set if and only if it is a psd matrix.
- (2) \mathbf{A} is a centering Gram matrix if and only if it is a centering psd matrix.
- (3) \mathbf{A} is a Euclidean square-distance matrix if and only if $-\frac{1}{2}\mathbf{A}^c$ is a centering psd matrix.

Proof. The first and the second statements are trivial. We prove the third one. Write $\mathbf{G} = -\frac{1}{2}\mathbf{A}^c$. If \mathbf{G} is a centering psd matrix, by (6.5), there is an $m \times n$ centered matrix $\mathbf{V} = [\mathbf{v}_1 \cdots \mathbf{v}_n]$ with $m \leq n$ such that

$$\mathbf{G} = \mathbf{V}'\mathbf{V}.$$

By Theorem 6.1, the matrix \mathbf{A} , defined by $\mathbf{A}^c = -2\mathbf{G}^c$, is a Euclidean square-distance matrix. On the other hand, if \mathbf{A} is the Euclidean square-distance matrix of a data set \mathcal{X} , then $\mathbf{G}^c = -\frac{1}{2}\mathbf{A}^c$ is the centering Gram matrix of \mathcal{X} so that it is a centering psd matrix. \square

Theorem 6.2 is important, for it characterizes Euclidean distance matrix and Gram matrix without using the information of the underlying data set.

We remark that Theorem 6.2 is essentially equivalent to a classical result of Young and Householder [6].

6.3 Description of Classical Multidimensional Scaling

6.3.1 CMDS Method Description

Let $\mathbf{D} = [d_{ij}]_{i,j=1}^n$ be a given distance matrix of a set of n objects, and $d > 0$ be an integer. The method of metric multidimensional scaling (MMDS) [7] is a procedure for finding a configuration $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\} \subset \mathbb{R}^d$, such that a certain distance matrix associated with \mathcal{Y} is as close as possible to the matrix \mathbf{D} , namely,

$$d_Y(\mathbf{y}_i, \mathbf{y}_j) \approx d_{ij}, \quad \forall i, j. \quad (6.8)$$

In practice, a lost function is adopted to measure the closeness in (6.8). The classical multidimensional scaling (CMDS) [8] adopts a Euclidean metric \mathbf{D} as its input. The following lemma plays the central role in CMDS.

Lemma 6.3. *Assume that an $n \times n$ matrix $\mathbf{D} = [d_{ij}]$ is a Euclidean metric and $\mathbf{S} = [d_{ij}^2]$ is the corresponding square-distance matrix. Let $\mathbf{G}^c = -\frac{1}{2}\mathbf{S}^c$. If the rank of \mathbf{G}^c is r , then there is an r -dimensional centered vector set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^r$ such that*

$$d_2(\mathbf{x}_i, \mathbf{x}_j) = d_{ij}, \quad 1 \leq i, j \leq n. \quad (6.9)$$

Proof. By Theorem 6.2, \mathbf{G}^c is a centering Gram matrix. Since the rank of \mathbf{G}^c is r , there exists an $r \times n$ centered data matrix \mathbf{X} such that $\mathbf{G}^c = \mathbf{X}'\mathbf{X}$. Then the centered data set \mathcal{X} satisfies (6.9). The lemma is proved. \square

We call r in Lemma 6.3 the intrinsic configuration dimension of \mathbf{D} and call \mathcal{X} the exact configuration of \mathbf{D} . If we want to use the configuration for visualization, and r is too large to meet the goal, then we seek for a low-dimensional configuration, say, a d -dimensional configuration \mathcal{Y} with $d \ll r$. Let both \mathbf{Y} and \mathbf{X} be treated as random vectors. Intuitively, \mathbf{Y} ought to be d principal components of \mathbf{X} . Hence, the lost function of CMDS is set to be

$$\eta(\mathcal{Y}) = \sum_{i,j=1}^n (d_{ij}^2 - d_2^2(\mathbf{y}_i, \mathbf{y}_j)), \quad \text{s.t. } \mathcal{Y} = T(\mathcal{X}),$$

where T is an o.g. project from \mathbb{R}^r to a d -subspace $S_d \subset \mathbb{R}^r$ and \mathcal{X} is an exact configuration of \mathbf{D} . Then the configurative point set \mathcal{Y} is the solution

of the minimization problem

$$\mathbf{Y} = \arg \min_{Y \in \mathfrak{M}_{d,n}} \eta(\mathcal{Y}), \quad \text{s.t. } \mathcal{Y} = T(\mathcal{X}). \quad (6.10)$$

To simplify the minimization problem (6.10), we establish some lemmas.

Lemma 6.4. *Let $\mathcal{Z} \subset \mathbb{R}^r$ be a given data set with corresponding Euclidean square-distance matrix $\mathbf{S}_Z = [s_{ij}]$, where $s_{ij} = d_2^2(\mathbf{z}_i, \mathbf{z}_j)$, and let \mathbf{G}_Z^c be its associated centering Gram matrix. Then*

$$\text{tr}(\mathbf{G}_Z^c) = \frac{1}{2n} \sum_{i=1}^n \sum_{j=1}^n s_{ij}. \quad (6.11)$$

Proof. Write $s_S = \sum_{i=1}^n \sum_{j=1}^n s_{ij}$. Let \mathbf{H} be the n -centralizing matrix and \mathbf{E} be the $n \times n$ matrix, in which all entries are 1. Let $\widehat{\mathcal{Z}} = \{\hat{\mathbf{z}}_i\}_{i=1}^n$ be the centered data set of \mathcal{Z} . By Theorem 6.2, we have

$$\mathbf{G}_Z^c = -\frac{1}{2} \mathbf{S}_Z^c = -\frac{1}{2} \mathbf{H} \mathbf{S}_Z \mathbf{H} = -\frac{1}{2} \left(\mathbf{S}_Z - \frac{1}{n} \mathbf{E} \mathbf{S}_Z - \frac{1}{n} \mathbf{S}_Z \mathbf{E} + \frac{1}{n^2} \mathbf{E} \mathbf{S}_Z \mathbf{E} \right),$$

which yields, by $s_{ii} = 0$,

$$\begin{aligned} \langle \hat{\mathbf{z}}_i, \hat{\mathbf{z}}_i \rangle &= -\frac{1}{2} \left(s_{ii} - \frac{1}{n} \sum_{k=1}^n s_{ik} - \frac{1}{n} \sum_{k=1}^n s_{kj} + \frac{1}{n^2} \sum_{k=1}^n \sum_{l=1}^n s_{kl} \right) \\ &= \frac{1}{n} \sum_{k=1}^n s_{ik} - \frac{1}{2n^2} s_S. \end{aligned}$$

Therefore,

$$\begin{aligned} \text{tr}(\mathbf{G}_Z^c) &= \sum_{i=1}^n \langle \hat{\mathbf{z}}_i, \hat{\mathbf{z}}_i \rangle = \sum_{i=1}^n \left(\frac{1}{n} \sum_k s_{ik} - \frac{1}{2n^2} s_S \right) \\ &= \frac{1}{n} \sum_{i=1}^n \sum_{k=1}^n s_{ik} - \frac{1}{2n} s_S \\ &= \frac{1}{2n} s_S. \end{aligned}$$

The proof of the lemma is completed. \square

Lemma 6.4 immediately yields the following.

Lemma 6.5. *Let $\mathbf{D}_Z = [d_2(\mathbf{z}_i, \mathbf{z}_j)]_{i,j=1}^n$ and $\hat{\mathbf{Z}} = [\hat{\mathbf{z}}_1, \dots, \hat{\mathbf{z}}_n]$. Then*

$$\|\hat{\mathbf{Z}}\|_F = \frac{1}{\sqrt{2n}} \|\mathbf{D}_Z\|_F.$$

Proof. Since $\|\mathbf{D}_Z\|_F^2 = \sum_{i=1}^n \sum_{j=1}^n s_{ij}$ and $\|\hat{\mathbf{Z}}\|_F^2 = \text{tr}(\mathbf{G}_Z^c)$, the result is a consequence of Lemma 6.4. \square

We now establish the main result in this section.

Theorem 6.3. *Let $\mathcal{X} \subset \mathbb{R}^r$ be the configuration of \mathbf{D} in Lemma 6.3 (where \mathcal{X} is centered) and the SVD of \mathbf{X} be given by*

$$\mathbf{X} = \mathbf{V} \boldsymbol{\Sigma}_r \mathbf{U}',$$

where $\boldsymbol{\Sigma}_r = \text{diag}(\sigma_1, \dots, \sigma_r) \in \mathfrak{D}_r$, $\mathbf{U} \in \mathfrak{O}_{n,r}$, and $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_r] \in \mathfrak{O}_r$. For a given $d \leq r$, let $\mathbf{V}_d = [\mathbf{v}_1, \dots, \mathbf{v}_d]$ and $\mathbf{Y} = \mathbf{V}'_d \mathbf{X}$. Then \mathbf{Y} is a solution of the minimization problem in (6.10) with

$$\eta(\mathcal{Y}) = \sum_{i=d+1}^r \sigma_i^2. \quad (6.12)$$

Proof. Let S_d be a d subspace of \mathbb{R}^r and $\mathbf{B} \in \mathfrak{O}_{r,d}$ be the matrix whose columns form an o.n. basis on S_d . Then $\mathbf{B}\mathbf{B}'$ represents the corresponding o.g. projection from \mathbb{R}^r to S_d . It is clear that

$$\sum_{i,j=1}^n d_2^2(i, j) - d_2^2(\mathbf{B}'\mathbf{x}_i, \mathbf{B}'\mathbf{x}_j) \leq \sum_{i,j=1}^n |\|\mathbf{x}_i - \mathbf{x}_j\|^2 - \|T\mathbf{x}_i - T\mathbf{x}_j\|^2|$$

and

$$\|(\mathbf{I} - \mathbf{B}\mathbf{B}')(\mathbf{x}_i - \mathbf{x}_j)\|^2 = d_2^2(i, j) - d_2^2(\mathbf{B}'\mathbf{x}_i, \mathbf{B}'\mathbf{x}_j).$$

Therefore, the minimization problem (6.10) is to find a $\mathbf{B}_* \in \mathfrak{O}_{r,d}$ such that the vector set $\mathbf{B}'_* \mathcal{X}$ minimizes $\eta(\mathcal{X})$:

$$\mathbf{B}_* = \arg \min_{\mathbf{B} \in \mathfrak{O}_{r,d}} \sum_{i,j=1}^n \|(\mathbf{I} - \mathbf{B}\mathbf{B}')(\mathbf{x}_i - \mathbf{x}_j)\|^2. \quad (6.13)$$

Writing $\hat{\mathbf{Z}} = (\mathbf{I} - \mathbf{B}\mathbf{B}')\mathbf{X}$, by Lemma 6.5, we have

$$\|\mathbf{D}_{\hat{\mathbf{Z}}}\|_F^2 = 2n\|\hat{\mathbf{Z}}\|_F^2.$$

Recall that

$$\eta(\hat{\mathcal{X}}) = \sum_{i,j=1}^n |d_{ij}^2 - d_2^2(\mathbf{B}\mathbf{B}'\mathbf{z}_i, \mathbf{B}\mathbf{B}'\mathbf{z}_j)| = \|\mathbf{D}_Z\|_F^2$$

and

$$\|\hat{\mathbf{Z}}\|_F = \|\mathbf{X} - \mathbf{B}_*(\mathbf{B}_*)'\mathbf{X}\|_F.$$

By Theorem 5.1 in Chapter 5, we have that the matrix

$$\mathbf{B}_* = \mathbf{U}_d$$

is the solution of (6.13), which yields that the set

$$\mathcal{Y} = \mathbf{U}'_d \mathbf{X}$$

is the solution of (6.10). Finally, the error estimate (6.12) is derived from

$$\eta(U_d \mathcal{X}) = \|\mathbf{X} - \mathbf{U}_d \mathbf{U}'_d \mathbf{X}\|_F^2 = \sum_{i=d+1}^r \sigma_i^2.$$

The proof is completed. \square

6.3.2 Relation between PCA and CMDS

The motivations and data models of PCA and CMDS are different: In PCA, a data set \mathcal{X} is given and the purpose of PCA is to find its leading d principal components that preserve the maximal variance. In CMDS, an inter-point distance matrix is given on a set of n objects and the purpose is to find a configurative point set in a low-dimensional Euclidean space that preserves the maximal similarities. However, if the set of objects in CMDS is digitized to a D -dimensional set \mathcal{X} in \mathbb{R}^D and the inter-point distance matrix is the Euclidean metric of \mathcal{X} , then PCA and CMDS become identical.

6.3.3 Weighted Graphic Description of CMDS

Assume that the node set of a graph is a set of n objects $N = [o_1, \dots, o_n]$ and the weight matrix of the graph is $\mathbf{W}(N)$. Then the weighted graph can be written as $G_N = [N, \mathbf{W}(N)]$. In CMDS, the weight matrix $\mathbf{W}(N)$ is an $n \times n$ Euclidean metric, representing the similarities between the objects in N . Since the distances of all pairs are measured, the graph G_N is a complete one. From the viewpoint of graph, CMDS is a method to find a new graph $G_Y = [\mathcal{Y}, \mathbf{W}(\mathcal{Y})]$, where $\mathcal{Y} \subset \mathbb{R}^d$ is a low-dimensional set, and $\mathbf{W}(\mathcal{Y})$ is the best approximation of $\mathbf{W}(N)$ in the sense that $\mathbf{W}(\mathcal{Y})$ minimizes the lost function $\|\mathbf{W}(N) - \mathbf{W}(\mathcal{Y})\|_F^2$.

6.4 CMDS Algorithm

The CMDS algorithm is relatively simple. Assume that the similarities of a set of n objects are given by the matrix \mathbf{D} , which is symmetric and Euclidean. The output of the algorithm is a d -dimensional configuration of \mathbf{D} . The algorithm consists of the following steps.

Step 1. Create centering Gram matrix. Apply the formula (6.7) to create a centering Gram matrix of the configuration: $\mathbf{G} = -\frac{1}{2}\mathbf{H}(\mathbf{D}^2)\mathbf{E}$, where \mathbf{D}^2 is the point-wise square of \mathbf{D} , also called the Hadamard square of \mathbf{D} .

Step 2. Make spectral decompose of \mathbf{G} . Assume that the rank of \mathbf{G} is r . Let the spectral decomposition of \mathbf{G} be $\mathbf{G} = \mathbf{U}\Lambda\mathbf{U}'$, where $\mathbf{U} = [\mathbf{u}_1 \cdots \mathbf{u}_r]$, and $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_r)$, with $\lambda_1 \geq \dots \geq \lambda_r$.

Step 3. Find configuration. Set $\mathbf{U}_d = [\mathbf{u}_1 \cdots \mathbf{u}_d]$ and $\Sigma_d = \text{diag}(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_d})$. Then the configuration is $\mathbf{Y} = \Sigma_d \mathbf{U}_d'$.

Remark. Let $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$ be two matrices having the same dimensions. The Hadamard product of \mathbf{A} and \mathbf{B} is the matrix $\mathbf{A} \boxtimes \mathbf{B} = [a_{ij}b_{ij}]$. Similarly, the Hadamard square of \mathbf{A} is the matrix $\mathbf{A}^2 = \mathbf{A} \boxtimes \mathbf{A}$. We often simply call them point-wise product and point-wise square.

The Matlab code of CMDS is presented in the following.

```
function [Y, ev] = drcmds(X, ndims, options)

% Classical Multidimensional Scaling
%
% SYNTAX:
%   Y = Y = DRCMDS(D, ndims, options);
%
% INPUTS:
%   'X': N x N symmetric matrix.
%   ndims: Dimension of output data.
%
% OPTIONS:
%   .verb: display the processing verbally.
%
% OUTPUT:
%   Y: d x N dimension data matrix.
%   ev: the corresponding eigenvalues.
%
% Example: Find relative locations of
%   500 cities according to the flight mileage.
%   Input: D= 500 x 500 symmetric
%   matrix with D(i,j) being the flight
%   mileage between City i and City j;
%   d=2;
%   options.null=0;
%   [Y, ev] = DRCMDS(X, d, options);
%
% Initialize options.
if nargin<3
    options=null;
end
if isfield(options, 'verb')
    verb = option.verb;
else
    verb = 1;
end
```

```
% check the input matrix
[N,M] =size(X);
if N~=M
    error('Input matrix is not squared.');
end
if any(any(X ~= X'))
    mes1=sprintf('Input matrix is not symmetric.\n');
    mes2='We use 1/2*(X+X') instead.';
    warning([mes1,mes2]);
    X = 1/2*(X+X');
end
if any(diag(X))
    mes1=sprintf('Diagonal does not vanish.\n');
    mes2='We force the diagonal vanish.';
    warning([mes1,mes2]);
    X=X-diag(diag(X));
end
% Part 1: Construct centering Gram matrix.
if verb
    disp('- construct Gram matrix');
end
DG = X.^2;
GC = -.5*(DG - sum(DG) * ones(1,N)/N ...
    - ones(N,1)*sum(DG)/N + sum(DG(:))/(N^2));
% Part 2: Kernel decomposition.
if verb
    disp('- Output low-dimensional data');
end
opt.disp = 0; opt.isreal = 1;
opt.issym = 1;
[Y, ev] = eigs(GC, ndims, 'LA', opt);
for i=1:ndims
    Y(:,i) = Y(:,i)*sqrt(val(i,i));
end
Y = Y';
```

References

- [1] Young, F.W.: Encyclopedia of Statistical Sciences, vol. 5, chap. Multidimensional scaling. John Wiley & Sons, Inc (1985).
- [2] Torgerson, W.S.: Multidimensional scaling I. Theory and method. Psychometrika 17, 401–419 (1952).
- [3] Young, F.W., Hamer, R.M.: Theory and Applications of Multidimensional Scaling. Erlbaum Associates, Hillsdale, NJ. (1994).
- [4] Shepard, R.: The Analysis of proximities: Multidimensional scaling with an unknown distance function I. Psychometrika 27, 125–140 (1962).
- [5] Shepard, R.: The Analysis of proximities: Multidimensional scaling with an

- unknown distance function II. *Psychometrika* 27, 219–246 (1962).
- [6] Young, G., Householder, A.: Discussion of a set of points in terms of their mutual distances. *Psychometrika* 3, 19–22 (1938).
- [7] Borg, I., Groenen, P.: Modern multidimensional scaling: Theory and applications. Springer-Verlag, New York (1997).
- [8] Cox, T.F., Cox, M.A.A.: Multidimensional Scaling. Chapman & Hall, London, New York (1994).

Chapter 7 Random Projection

Abstract Principal component analysis (PCA) is a very important linear method for dimensionality reduction. It measures data distortion globally by the Frobenius norm of the matrix of data difference. The reduced data of PCA consists of several leading eigenvectors of the covariance matrix of the data set. Hence, PCA may not preserve the local separation of the original data. To respect local properties of data in dimensionality reduction (DR), we employ Lipschitz embedding. Random projection is a powerful method to construct Lipschitz mappings to realize dimensionality reduction with a high probability. Random projection does not introduce a significant distortion when the dimension and cardinality of data both are large. It randomly projects the original high-dimensional data into a lower-dimensional subspace. Because the projection costs linear computational time, the method is computationally efficient, yet produces sufficient accuracy with a high probability. In Section 7.1, we give a review of Lipschitz embedding. In Section 7.2, we introduce random matrices and random projection algorithms. In Section 7.3, the justification of the validity of random projection is presented in detail. Particularly, Johnson and Lindenstrauss Lemma will be proved in this section. The applications of random projection are given in Section 7.4.

7.1 Introduction

7.1.1 Lipschitz Embeddings

In Chapters 5 and 6, we introduced principal component analysis and classical multidimensional scaling for dimensionality reduction, and proved their equivalence when the input is the data set. PCA employs the spectral decomposition of the covariance matrix of the input data set and the reduced data consists of several leading eigenvectors of the matrix. Hence, PCA is the

linear method that projects the original data into a subspace, measuring data distortion globally by a quadratic sum of the Euclidean differences between the original data and the DR data. It is clear that such a notion of distortion only captures a significant global property, but does not offer any local guarantees. As a result, the distance between a pair of points in the DR data set can be arbitrarily smaller than its corresponding pair in the original data set if that is advantageous to minimizing the global distortion. For example, let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ be the original (centered) data set and the singular value decomposition of the data matrix $\mathbf{X} = [\mathbf{x}_1 \dots \mathbf{x}_n]$ be

$$\mathbf{X} = \sum_{i=1}^r \sigma_i \mathbf{v}_i \mathbf{u}'_i, \quad r \leq D.$$

Then the reduced data $\mathcal{Y} \subset \mathbb{R}^d$ of PCA are represented by the matrix $\mathbf{Y} = [\mathbf{v}_1 \dots \mathbf{v}_d]' \mathbf{X}$. If $d < r$, all vectors in the subspace spanned by the set $\{\mathbf{v}_{d+1}, \dots, \mathbf{v}_r\}$ will be mapped to zero vector so that the distances between them cannot be preserved.

In mathematics, an embedding means a structure-preserving mapping. A dimensionality reduction method adopts a special mapping. In some applications, for example, in text document ordering, a dimensionality reduction method is required to preserve the separation of the objects of interest. Hence, studying the embedding that respects local properties is important. In the applications mentioned above, we need the embedding, which guarantees that distances between all pairs are approximately maintained. In mathematics, these embeddings are called Lipschitz embeddings, or Lipschitz mappings.

Definition 7.1. A mapping $f : \mathcal{X} \subset \mathbb{R}^D \rightarrow \mathbb{R}^k$ is called a Lipschitz mapping (on \mathcal{X}) if there are two positive constants A and B such that for each pair of vectors $\mathbf{u}, \mathbf{v} \in \mathcal{X}$, the following holds.

$$A\|\mathbf{u} - \mathbf{v}\|^2 \leq \|f(\mathbf{u}) - f(\mathbf{v})\|^2 \leq B\|\mathbf{u} - \mathbf{v}\|^2.$$

In the definition (7.1), the Euclidean norm is applied. But it can be replaced by other norms. Under a Lipschitz mapping, the distances between all pairs of vectors in the data set \mathcal{X} are approximately preserved.

7.1.2 JL-Embeddings

Many deep and beautiful results already exist in the study of Lipschitz mappings. However, to construct Lipschitz mappings for data reduction is not easy, particularly, in deterministic algorithms. Random projection proves a useful tool in the construction of Lipschitz mappings.

A random projection is a linear mapping that projects the data set into a random subspace. Assume that the data set \mathcal{X} is a subset of the Euclidean

space \mathbb{R}^D . In Chapter 5, we considered \mathcal{X} as a sample set of a D -dimensional random vector $\mathbf{X} = [X_1, \dots, X_D]'$. Let $\mathcal{R} = \{\mathbf{r}_1 \dots \mathbf{r}_k\} \subset \mathbb{R}^D$ be a set of k ($k < D$) random vectors and $\mathbf{R} = [\mathbf{r}_1 \dots \mathbf{r}_k]$ be the corresponding random matrix. The mapping $f_r : \mathcal{X} \rightarrow \mathcal{Y} \subset \mathbb{R}^k, \mathbf{Y} = \mathbf{R}'\mathbf{X}$ is called a random projection. The relation $\mathbf{Y} = \mathbf{R}'\mathbf{X}$ indicates that \mathbf{Y} is a random vector sampled in a k -subspace of \mathbb{R}^D . The feasibility of random projection algorithms is based on the result of Johnson and Lindenstrauss [1] called Johnson-Lindenstrauss Lemma, which will be studied in the next section. Hence, the Lipschitz embeddings generated by random projections are often called JL-embeddings. The algorithms of JL-embeddings were first developed in [2] and have been successfully used in solving a variety of problems. Currently, JL-embeddings become an important part of modern algorithmic design. In this chapter, we shall introduce the random projection algorithms and present their justification.

Random projection also plays an important role in matrix decomposition. In many applications, singular value decomposition (SVD) of matrices plays an important role. For example, PCA method employs SVD to find principal components. When a data set is in a very high-dimensional space and the set contains a large number of points, computing SVD of the data matrix becomes very computationally expensive and unstable. For instance, if a data set $\mathcal{X} \subset \mathbb{R}^D$ has n points (i.e., $|\mathcal{X}| = n$) and $D \sim n$, then SVD of \mathbf{X} requires $O(n^3)$ operations and $O(n^2)$ memory space. In many applications, such as market basket data analysis, text document ordering, and image processing, the data sets are of this type. Hence, directly computing SVD of the data matrices in these applications often becomes unfeasible. Random projection maps the data into a k -dimensional subspace ($k \ll D$), which “nearly” catches the required principal components. The mapping only costs $O(n)$ if $k \ll n$. Hence, random projection method is computationally efficient and simple, yet produces sufficient accuracy with a high probability. We will discuss the randomized SVD in Section 15.2.

Random projection is a kind of Monte Carlo method, while PCA is a deterministic algorithm. Random projection is employed to replace PCA when the deterministic algorithms become unfeasible or impossible to compute required results.

7.2 Random Projection Algorithms

7.2.1 Random Matrices and Random Projection

Random projection is realized by random matrix, which is introduced in this section. A random variable r is denoted by $r \sim N(0, 1)$ if its probability

density function (pdf) f_r is the Gaussian probability distribution $N(0, 1)$:

$$f_r(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}.$$

It is denoted by $r \sim U(\{-1, 1\})$ if it is uniformly distributed on $\{-1, 1\}$:

$$r := \begin{cases} 1, & \text{with probability } 1/2, \\ -1, & \text{with probability } 1/2; \end{cases}$$

and it is denoted by $r \sim U(\{\pm\sqrt{3}, 0\})$ if

$$r := \begin{cases} \sqrt{3}, & \text{with probability } 1/6, \\ 0, & \text{with probability } 2/3, \\ -\sqrt{3}, & \text{with probability } 1/6. \end{cases} \quad (7.1)$$

For convenience, we shall call $r \sim N(0, 1)$, $\sim U(\{-1, 1\})$, and $\sim U(\{\pm\sqrt{3}, 0\})$ the random variables of Type-1, Type-2, and Type-3, respectively. It can be verified that each of these random variables has zero mean and unit variance: $E(r) = 0$, $\text{Var}(r) = 1$.

A random matrix is a matrix such that all of its entries are independent and identically distributed (i.i.d.) random variables. To classify various random matrices, we introduce the following.

Definition 7.2. Let $\mathbf{R} = [r_{i,j}]$ be a random matrix of independent random variables $r_{i,j}$. Then \mathbf{R} is called a random matrix of Type-1 (or Gaussian), Type-2, or Type-3, if $r_{i,j}$ is of Type-1, Type-2, or Type-3, respectively.

In the following, without being specific, a random matrix may belong to any of the above three types. The notion of random projection (RP) can be described as follows.

Definition 7.3. Let $\mathbf{R} = [r_{i,j}]$ be a $k \times D$ random matrix of a certain type. A normalized random projection $R : \mathbb{R}^D \rightarrow \mathbb{R}^k$ is the projection that maps a D -dimensional vector \mathbf{u} to a k -dimensional vector \mathbf{u}' , defined by

$$\mathbf{u}' = R(\mathbf{u}) = \frac{1}{\sqrt{k}} \mathbf{R}\mathbf{u}. \quad (7.2)$$

Since the random matrices of Type-2 and Type-3 are simpler than those of Type-1, they accelerate computing speed. Note also that the random matrices of Type-3 are sparse. The random projection method provides a very simple strategy for random algorithm development. First, the data set is randomly projected into a lower dimensional space. Then, any desirable algorithm is applied to the transformed low-dimensional data set. Since random projection preserves data features with high probability, this simple protocol has been followed to successfully design learning algorithms [3–6] and other algorithms.

7.2.2 Random Projection Algorithms

A random projection algorithm is extremely simple. It has only two parts: random matrix creation and matrix multiplication. The following is an algorithm for random matrix creation.

```
function rm = randmtr(n,m, type)
% Create n x m Random matrix of type 1-3.
%
% SYNTAX: rm = RANDMTR(n,m, type)
% INPUTS
%   N: Row number of random matrix.
%   M: Column number of random matrix.
%   TYPE: Random type of matrix.
% OUTPUT
%   RM: n x m random matrix of Type 1, 2, or 3.
% Example: Create a 1000x800 random matrix of Type-1.
%   rm = RANDMTR(1000, 800, 1);
switch type
    case 1
        rm = randn(n,m);
    case 2
        A = ( rand(n,m)>0.5 );
        rm = A- (~A);
    case 3
        A=rand(n,m);
        rm = sqrt(3)*((A>1/6)-(A<1/6));
end
```

The Matlab code for random projection is also very simple. The algorithm we developed runs for two purposes. It simply performs a random projection if the parameter `needsvd` is set to 0. Otherwise (i.e., the parameter `needsvd` is set to 1), it adds a complete SVD to the reduced data matrix. The algorithm calls the m-function `randmtr`.

```
function [Y, varargout]= rdmprj(X, k, type, needsvd)
% Perform random projection.
%
% SYNTAX: Y = RNDPRJ(X,k,type,0);
%          Y = RNDPRJ(X,k,type,1);
%          [Y,ev]= RNDPRJ(X,k,type,1);
% INPUTS
%   X: Data matrix, of which each column is a point.
%   K: Dimension of reduced data.
%   TYPE: Type of random matrix.
%   NEEDSVD: Choose 1 if SVD is needed
%             for reduced data.
% OUTPUT
%   Y: Reduced data matrix.
%   ev: Eigenvalue set. The output is only available
```

```
%      when SVD is required for reduced data.
% Example: Randomly map a 800x1000 data matrix
%           X to 7-D using random matrix of type 3.
%           Y = RNDPRJ(X,7,3,0);
d = size(X,1);
R = randmtr(d, k, type);
Y = 1/sqrt(k)*R*X;
if needsvd
    [v, ev] = svd(Y);
    Y = v'*Y;
    varargout1 = diag(ev)';
end
```

7.3 Justification

In this section, we justify the validity of random projection for dimensionality reduction. Recall that random projection is used to realize Lipschitz embedding. Note that, when $k < D$, any projection $f : \mathbb{R}^D \rightarrow \mathbb{R}^k$ has a non-trivial null space. Therefore, it cannot be a Lipschitz embedding. Hence, Lipschitz embeddings only exist for certain subsets. On a given subset the existence of Lipschitz embeddings needs to be justified. Johnson and Lindenstrauss are pioneers in the study of this direction.

7.3.1 Johnson and Lindenstrauss Lemma

Johnson and Lindenstrauss in their pioneering paper [1] gave the following lemma, which reveals the dependency between the dimension k (of the space, into which data embedded,) and the cardinality of the data set \mathcal{X} .

Lemma 7.1. *Let $\varepsilon > 0$ and integer n be given. Then for all positive integers $k \geq k_0 = O(\varepsilon^{-2} \log n)$ and a set \mathcal{X} of n points, which are randomly selected in \mathbb{R}^D , there exists a projection $f : \mathbb{R}^D \rightarrow \mathbb{R}^k$, which satisfies*

$$(1 - \varepsilon)\|\mathbf{u} - \mathbf{v}\|^2 \leq \|f(\mathbf{u}) - f(\mathbf{v})\|^2 \leq (1 + \varepsilon)\|\mathbf{u} - \mathbf{v}\|^2, \quad (7.3)$$

for all $\mathbf{u}, \mathbf{v} \in \mathcal{X}$.

The projection f in Lemma 7.1 is called the JL-embedding. In the original paper [1], the range of k is not given by an explicit formula. Various extensions of Lemma 7.1 were established over the past two decades. It has also been pointed out that JL-embeddings could be realized by random projections with positive probabilities, and the explicit formula of the lower bound of k in Lemma 7.1 has been given. For instance, Frankl and Maehara in their paper [7] gave an lower bound of $k: \lceil 9(\varepsilon^2 - 2\varepsilon^3/3)^{-1} \ln n \rceil + 1$, where the

notation $\lceil r \rceil$ denotes the ceiling integer of $r \in \mathbb{R}$. Indyk and Motwani in [2], Dasgupta and Gupta in [8], and Achlioptas in [3] improved the lower bound of k to $4(\varepsilon^2/2 - \varepsilon^3/3)^{-1} \ln n$.

We first introduce a lemma, which partially explains why random projection can realize JL-embedding.

Lemma 7.2. *Let $\mathbf{R} = [r_{ij}]$ be a $k \times D$ matrix, in which all entries are i.i.d. random variables with 0 mean and unit variance. Let the mapping $f : \mathbb{R}^D \rightarrow \mathbb{R}^k$ be defined by*

$$f(\mathbf{a}) = \frac{1}{\sqrt{k}} \mathbf{R}\mathbf{a}, \quad \mathbf{a} \in \mathbb{R}^D. \quad (7.4)$$

Then $f(\mathbf{a})$ is a random vector in \mathbb{R}^k , of which all components are i.i.d. random variables with 0 mean and variance $\frac{1}{k} \|\mathbf{a}\|^2$. Therefore,

$$\mathbb{E}(\|f(\mathbf{a})\|^2) = \|\mathbf{a}\|^2. \quad (7.5)$$

Proof. We denote the i th row vector of \mathbf{R} by \mathbf{r}^i and denote the inner product $\langle \mathbf{r}^i, \mathbf{a} \rangle$ by c_i . Then $f(\mathbf{a}) = \mathbf{c} = [c_1, \dots, c_k]'$. We have

$$\begin{aligned} \mathbb{E}(c_i) &= \frac{1}{\sqrt{k}} \sum_{j=1}^D a_j \mathbb{E}(r_{ij}) = 0, \\ \mathbb{E}(c_i c_j) &= \frac{1}{k} \sum_{m=1}^D \sum_{l=1}^D a_l a_m \mathbb{E}(r_{il} r_{jm}) = 0, \quad i \neq j, \end{aligned}$$

and

$$\begin{aligned} \mathbb{E}(c_i^2) &= \mathbb{E} \left(\left(\sum_{j=1}^D a_j (r_{ij}) \right)^2 \right) \\ &= \frac{1}{k} \left(\sum_{j=1}^D a_j^2 \mathbb{E}(r_{ij}^2) + 2 \sum_{l \neq m} a_l a_m \mathbb{E}(r_{il}) \mathbb{E}(r_{im}) \right) \\ &= \frac{1}{k} \sum_{j=1}^D a_j^2, \end{aligned}$$

which yields that all entries of $f(\mathbf{a})$ are i.i.d. random variables with 0 mean and variance $\frac{1}{k} \|\mathbf{a}\|^2$. Finally, we have

$$\mathbb{E}(\|f(\mathbf{a})\|^2) = \sum_{i=1}^k \mathbb{E}(c_i^2) = k \frac{\|\mathbf{a}\|^2}{k} = \|\mathbf{a}\|^2.$$

The lemma shows that the random projection f has the properties $E(\|f(\mathbf{a})\|^2) = \|\mathbf{a}\|^2$ for any independent matrix $[r_{ij}]$ with $E(r_{ij}) = 0$ and $\text{Var}(r_{ij}) = 1$. Since the random matrices of Types 1–3 have such properties, they are good candidates for the construction of JL-embedding.

From Lemma 7.2, we have:

Corollary 7.1. *Let f be the mapping defined in Lemma 7.2 and $\mathbf{a} \in \mathbb{R}^D$ be a unit vector. Then $\sqrt{k}f(\mathbf{a})$ is a k -dimensional random vector whose entries are i.i.d. random variables with 0 mean and unit variance.*

Note that $\sqrt{k}f(\mathbf{a}) = \mathbf{R}\mathbf{a}$. The corollary claims that if a $k \times D$ random matrix \mathbf{R} whose entries are i.i.d. random variables with 0 mean and unit variance, and $\mathbf{a} \in \mathbb{R}^D$ is a unit vector, then the entries of the random vector $\mathbf{R}\mathbf{a}$ are also i.i.d with zero mean and unit variance.

7.3.2 Random Projection based on Gaussian Distribution

We now introduce the result proved by Dasgupta and Gupta in [8], which states that if the random matrix in the random projection (7.2) is of Gaussian type, then the random projection realizes a JL-embedding for a certain dimension k .

Theorem 7.1. *For any $0 < \varepsilon < 1$ and any integer $n > 0$, let k be a positive integer such that*

$$k \geq 4(\varepsilon^2/2 - \varepsilon^3/3)^{-1} \ln n. \quad (7.6)$$

Then for any set $\mathcal{V} \subset \mathbb{R}^D$ of n points, there is a linear mapping $f : \mathbb{R}^D \rightarrow \mathbb{R}^k$, such that for all $\mathbf{u}, \mathbf{v} \in \mathcal{V}$,

$$(1 - \varepsilon)\|\mathbf{u} - \mathbf{v}\|^2 \leq \|f(\mathbf{u}) - f(\mathbf{v})\|^2 \leq (1 + \varepsilon)\|\mathbf{u} - \mathbf{v}\|^2. \quad (7.7)$$

Furthermore, this mapping can be found in randomized polynomial time.

Remark 7.1. An algorithm has randomized polynomial time if it runs in polynomial time in the input size and, if the correct answer is NO, it always returns NO, while if the correct answer is YES, then it returns YES with a positive (constant) probability.

To prove the theorem, we need:

Lemma 7.3. *Let \mathbf{R} be a $k \times D$ ($k \leq D$) random matrix of Gaussian type and $\mathbf{a} \in \mathbb{R}^D$ be a unit vector. Let $\mathbf{y} = \mathbf{R}\mathbf{a}$ and $1 < \beta$. Then*

$$\Pr \left[\|\mathbf{y}\|^2 \leq \frac{k}{\beta} \right] < \exp \left(\frac{k}{2} \left(1 - \frac{1}{\beta} - \ln \beta \right) \right), \quad (7.8)$$

$$\Pr \left[\|\mathbf{y}\|^2 \geq k\beta \right] < \exp \left(\frac{k}{2} (1 - \beta + \ln \beta) \right). \quad (7.9)$$

Proof. We first prove (7.8). For any $h > 0$,

$$\Pr \left[\|\mathbf{y}\|^2 \leq \frac{k}{\beta} \right] = \Pr \left[\exp(-h\|\mathbf{y}\|^2) \geq \exp \left(-\frac{hk}{\beta} \right) \right]. \quad (7.10)$$

Applying Markov's inequality, we have

$$\Pr \left[\exp(-h\|\mathbf{y}\|^2) \geq \exp \left(-\frac{hk}{\beta} \right) \right] \leq E(\exp(-h\|\mathbf{y}\|^2)) \exp \left(\frac{hk}{\beta} \right). \quad (7.11)$$

Note that $y_i = \sum_{j=1}^D a_j r_{ij}$, where $r_{ij} \sim N(0, 1)$. Hence, y_i is distributed normally. By Corollary 7.1, $y_i \sim N(0, 1)$, $1 \leq i \leq k$. Hence, we obtain

$$E(\exp(-hy_i^2)) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \exp(-\lambda^2/2 - h\lambda^2) d\lambda = \frac{1}{\sqrt{1+2h}}, \quad (7.12)$$

which yields

$$E(\exp(-h\|\mathbf{y}\|^2)) \exp \left(\frac{hk}{\beta} \right) = (1+2h)^{-k/2} \exp \left(\frac{hk}{\beta} \right). \quad (7.13)$$

The function

$$g(h) = (1+2h)^{-k/2} \exp \left(\frac{hk}{\beta} \right), \quad 0 < h,$$

has its minimum at $h^* = \frac{\beta-1}{2}$ and the minimal value is

$$g(h^*) = \beta^{-k/2} \exp \left(\frac{k}{2} \left(1 - \frac{1}{\beta} - \ln \beta \right) \right).$$

Hence, (7.8) holds. To prove (7.9), we employ

$$\begin{aligned} \Pr [\|\mathbf{y}\|^2 \geq k\beta] &= \Pr [\exp(h\|\mathbf{y}\|^2) \geq \exp(hk\beta)] \\ &\leq E(\exp(h\|\mathbf{y}\|^2)) \exp(-hk\beta). \end{aligned}$$

Similar to (7.12), we have $E(\exp(h\|\mathbf{y}\|^2)) = \frac{1}{\sqrt{1-2h}}$, ($0 < h < 1/2$), which yields

$$\Pr [\|\mathbf{y}\|^2 \geq k\beta] \leq (1-2h)^{-k/2} \exp(-hk\beta).$$

The minimum of the function $\tilde{g}(h) \stackrel{\text{def}}{=} (1-2h)^{-k/2} \exp(-hk\beta)$ is obtained at $\bar{h} = \frac{1-1/\beta}{2}$, and the minimal value is

$$\tilde{g}(\bar{h}) = \beta^{k/2} \exp \left(-\frac{k}{2}(\beta-1) \right) = \exp \left(\frac{k}{2} (1-\beta + \ln \beta) \right).$$

Hence, Equation (7.9) holds and the lemma is proved. \square

We return to the proof of Theorem 7.1.

Proof of Theorem 7.1. Let \mathbf{R} be a $k \times D$ random matrix whose entries are i.i.d. random variables $\sim N(0, 1)$. We define $f : \mathbb{R}^D \rightarrow \mathbb{R}^k$ as in (7.4). For each pair of \mathbf{u} and \mathbf{v} in \mathcal{V} ($\mathbf{u} \neq \mathbf{v}$), we set

$$\mathbf{a} = \frac{\mathbf{u} - \mathbf{v}}{\|\mathbf{u} - \mathbf{v}\|}, \quad z = f(\mathbf{a}) \quad \mathbf{y} = \sqrt{k}z.$$

Then

$$\begin{aligned} \Pr\left(\frac{\|f(\mathbf{u}) - f(\mathbf{v})\|^2}{\|(\mathbf{u} - \mathbf{v})\|^2} \leqslant (1 - \varepsilon)\right) &= \Pr(\|z\|^2 \leqslant (1 - \varepsilon)) \\ &= \Pr(\|\mathbf{y}\|^2 \leqslant (1 - \varepsilon)k). \end{aligned}$$

Similarly,

$$\Pr\left(\frac{\|f(\mathbf{u}) - f(\mathbf{v})\|^2}{\|(\mathbf{u} - \mathbf{v})\|^2} \geqslant (1 + \varepsilon)\right) = \Pr(\|\mathbf{y}\|^2 \geqslant (1 + \varepsilon)k).$$

Applying Lemma 7.3 to \mathbf{y} with $\frac{1}{\beta} = 1 - \varepsilon$, we have

$$\begin{aligned} \Pr(\|\mathbf{y}\|^2 \leqslant (1 - \varepsilon)k) &< \exp\left(\frac{k}{2}(1 - (1 - \varepsilon) + \ln(1 - \varepsilon))\right) \\ &\leqslant \exp\left(\frac{k}{2}\left(1 - (1 - \varepsilon) - \left(\varepsilon - \frac{\varepsilon^2}{2}\right)\right)\right) \\ &= \exp\left(-\frac{k\varepsilon^2}{4}\right) \leqslant \exp(-2 \ln n) = \frac{1}{n^2}, \end{aligned}$$

where the inequality $\ln(1 - \varepsilon) \leqslant -\varepsilon - \varepsilon^2/2$ is used for getting the second line, and the condition (7.6) is applied to deriving the inequality in the last line. Similarly, we have

$$\begin{aligned} \Pr(\|\mathbf{y}\|^2 \geqslant (1 + \varepsilon)k) &< \exp\left(\frac{k}{2}(1 - (1 + \varepsilon) + \ln(1 + \varepsilon))\right) \\ &\leqslant \exp\left(\frac{k}{2}\left(-\varepsilon + \left(\varepsilon - \frac{\varepsilon^2}{2} + \frac{\varepsilon^3}{3}\right)\right)\right) \\ &= \exp\left(-\frac{k(\varepsilon^2/2 - \varepsilon^3/3)}{2}\right) \leqslant \exp(-2 \ln n) = \frac{1}{n^2}. \end{aligned}$$

Therefore, for each pair \mathbf{u} and \mathbf{v} in \mathcal{V} ,

$$\Pr\left(\frac{\|f(\mathbf{u}) - f(\mathbf{v})\|^2}{\|(\mathbf{u} - \mathbf{v})\|^2} \notin [1 - \varepsilon, 1 + \varepsilon]\right) < \frac{2}{n^2}.$$

Since the cardinality of \mathcal{V} is n , there are total $n(n-1)/2$ pairs. The probability of the event that at least one pair does not satisfy (7.1) is less than $1 - \frac{n(n-1)}{2} \frac{2}{n^2} = 1/n > 0$. The theorem is proved. \square

From the proof of Theorem 7.1, we claim that a random projection with i.i.d. $\sim N(0, 1)$ entries generates a JL-embedding. This is

Corollary 7.2. *Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ be a given data set, k be an integer satisfying (7.6), and \mathbf{R} be a $k \times D$ random matrix of Type-1. Let $\mathbf{Y} = \frac{1}{\sqrt{k}} \mathbf{R} \mathbf{X}$. Then for each pair \mathbf{x}_i and \mathbf{x}_j in \mathcal{X} , the following inequalities*

$$(1 - \varepsilon) \|\mathbf{x}_i - \mathbf{x}_j\|^2 \leq \|\mathbf{y}_i - \mathbf{y}_j\|^2 \leq (1 + \varepsilon) \|\mathbf{x}_i - \mathbf{x}_j\|^2$$

hold with the probability being at least $1 - 2/n^2$.

7.3.3 Random Projection based on Types 2 and 3

In the previous subsection, we proved that random projections of Gaussian type generate JL-embeddings. In this subsection, we prove that JL-embeddings can also be realized by the random projections of Types 2 and 3. This result was first obtained by Achlioptas [3].

Theorem 7.2. *Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ be an arbitrary data set and $\mathbf{X} = [\mathbf{x}_1 \ \dots \ \mathbf{x}_n]$ be corresponding $D \times n$ matrix. For a given $0 < \varepsilon < 1$, let k be any integer that satisfies (7.6) and \mathbf{R} be a $k \times D$ random matrix of Type-2 or Type-3. Let $f : \mathbb{R}^D \rightarrow \mathbb{R}^k$ be the mapping defined by*

$$\mathbf{u}' = \frac{1}{\sqrt{k}} \mathbf{R} \mathbf{u}, \quad \mathbf{u} \in \mathbb{R}^D.$$

Let $\mathbf{y}_i = f(\mathbf{x}_i)$, $1 \leq i \leq n$. Then the probability for f to satisfy the Lipschitz condition on \mathcal{X} :

$$(1 - \varepsilon) \|\mathbf{x}_i - \mathbf{x}_j\|^2 \leq \|\mathbf{y}_i - \mathbf{y}_j\|^2 \leq (1 + \varepsilon) \|\mathbf{x}_i - \mathbf{x}_j\|^2, \quad \mathbf{x}_i, \mathbf{x}_j \in \mathcal{X}, \quad (7.14)$$

is at least $1 - n^{-\beta}$, where

$$\beta = \frac{k(\varepsilon^2/4 - \varepsilon^3/6)}{\ln n} - 2 > 0.$$

The random variables of Type 2 (or Type 3) is different from Type 1 since the sum and difference of two independent random variables of Type 1 are still of Type 1, but it is not true for Type 2 or 3. To prove Theorem 7.2, we need some properties of the random variables of Types 2 and 3. Let G , B , and T denote the random variables of Types 1, 2, and 3 respectively. Let $\mathbf{G} = [G_1, \dots, G_D]'$, $\mathbf{B} = [B_1, \dots, B_D]'$, and $\mathbf{T} = [T_1, \dots, T_D]'$ denote the D -dimensional vectors whose entries are i.i.d. random variables of Type-1, Type-2, and Type-3, respectively. For a unit vector $\mathbf{a} \in \mathbb{R}^D$, we write $G_{\mathbf{a}} = \sum_{j=1}^D a_j G_j$, $T_{\mathbf{a}} = \sum_{j=1}^D a_j T_j$, and $B_{\mathbf{a}} = \sum_{j=1}^D a_j B_j$. We also employ the notation of multi-index $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_D]$, $\alpha_i \in \mathbb{Z}_+$, setting $|\boldsymbol{\alpha}| = \sum_{i=1}^D \alpha_i$.

and $\mathbf{X}^\alpha = X_1^{\alpha_1} X_2^{\alpha_2} \cdots X_D^{\alpha_D}$. Let $m \in \mathbb{Z}_+$ be a positive integer. Then $B_{\mathbf{a}}^m$ is a homogeneous polynomial of degree m with respect to $\{B_1, \dots, B_D\}$:

$$(B_{\mathbf{a}})^m = \sum_{|\alpha|=m} \mathbf{a}^\alpha \mathbf{B}^\alpha. \quad (7.15)$$

Similarly, $T_{\mathbf{a}}^m$ is a homogeneous polynomial of degree m with respect to $\{T_1, \dots, T_D\}$:

$$(T_{\mathbf{a}})^m = \sum_{|\alpha|=m} \mathbf{a}^\alpha \mathbf{T}^\alpha, \quad (7.16)$$

and $G_{\mathbf{a}}^m$ is a homogeneous polynomial of degree m with respect to $\{G_1, \dots, G_D\}$:

$$(G_{\mathbf{a}})^m = \sum_{|\alpha|=m} \mathbf{a}^\alpha \mathbf{G}^\alpha, \quad (7.17)$$

The following lemma can be proved by directly computing.

Lemma 7.4. *Let $l \in \mathbb{Z}_+$ and G, B , and T be the random variables of Types 1, 2, and 3, respectively. Then*

$$\mathbb{E}(G^{2l+1}) = \mathbb{E}(B^{2l+1}) = \mathbb{E}(T^{2l+1}) = 0, \quad (7.18)$$

$$\mathbb{E}(G^{2l}) = \frac{(2l)!}{l!2^l}, \quad \mathbb{E}(B^{2l}) = 1, \quad \mathbb{E}(T^{2l}) = 3^{l-1}. \quad (7.19)$$

Therefore,

$$\mathbb{E}(B^{2l}) < \mathbb{E}(G^{2l}), \quad \mathbb{E}(T^{2l}) < \mathbb{E}(G^{2l}). \quad (7.20)$$

We now generalize Lemma 7.4 to $(B_{\mathbf{a}})^m$ and $(T_{\mathbf{a}})^m$.

Lemma 7.5. *Let $B_{\mathbf{a}}$, $T_{\mathbf{a}}$, and $G_{\mathbf{a}}$ be defined by (7.15), (7.16), and (7.17), respectively. Then*

$$\mathbb{E}((B_{\mathbf{a}})^{2l+1}) = 0, \quad \mathbb{E}((B_{\mathbf{a}})^{2l}) < \mathbb{E}((G_{\mathbf{a}})^{2l}), \quad (7.21)$$

$$\mathbb{E}((T_{\mathbf{a}})^{2l+1}) = 0, \quad \mathbb{E}((T_{\mathbf{a}})^{2l}) < \mathbb{E}((G_{\mathbf{a}})^{2l}). \quad (7.22)$$

Proof. We prove (7.21) only, since the proof for (7.22) is similar. We have

$$(B_{\mathbf{a}})^{2l+1} = \sum_{|\alpha|=2l+1} \mathbf{a}^\alpha \mathbf{B}^\alpha.$$

In $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_D]' (|\boldsymbol{\alpha}| = 2l + 1)$, at least one index is odd. Without loss of generality, we assume that $\alpha_1 = 2s + 1, s \in \mathbb{Z}_+$. Then we have

$$\mathbb{E}(B^\alpha) = \mathbb{E}(B_1^{2s+1}) \mathbb{E}(B_2^{\alpha_2} \cdots B_D^{\alpha_D}).$$

By Lemma 7.4, $\mathbb{E}(B_1^{2s+1}) = 0$. Hence, $\mathbb{E}(\mathbf{B}^\alpha) = 0$ for all α with $|\alpha| = 2l + 1$. The first equation of (7.21) is proved. To prove the second equation in (7.21), we employ

$$\mathbb{E}((B_{\mathbf{a}})^{2l}) = \sum_{|\beta|=l} (\mathbf{a}^{2\beta} \mathbb{E}(\mathbf{B}^{2\beta})),$$

where all odd-power terms vanish because they have zero mean. By (7.20), $E(B_i^{2\beta_i}) < E(G_i^{2\beta_i})$ for each i , which yields

$$E(B^{2\beta}) = \prod_{i=1}^D E(B_i^{2\beta_i}) < \prod_{i=1}^D E(G_i^{2\beta_i}) = E(G^{2\beta}).$$

Hence,

$$E((B_a)^{2l}) < E((G_a)^{2l}).$$

The lemma is proved. \square

We now reach the following.

Lemma 7.6. *Let \mathbf{B} be a D -dimensional random vector of Type 2 or 3, and $\mathbf{a} \in \mathbb{R}^D$ be a unit vector. Let $z = \langle \mathbf{B}, \mathbf{a} \rangle$. Then for each $h > 0$,*

$$E(\exp(hz^2)) \leq \frac{1}{\sqrt{1-2h}}, \quad (7.23)$$

$$E(z^4) \leq \frac{3}{D}. \quad (7.24)$$

Proof. Let \mathbf{G} be a D -dimensional random vector of Type 1 and $y = \langle \mathbf{G}, \mathbf{a} \rangle$. By Lemma 7.5, we have

$$E(\exp(hz^2)) < E(\exp(hy^2)), \quad (7.25)$$

where the right-hand side is equal to $\frac{1}{\sqrt{1-2h}}$. Equation (7.23) is proved. By Lemma 7.4, if the random vector \mathbf{B} is of Type 3, then

$$E(z^4) = \sum_{i=1}^D a_i^4 + \sum_{i \neq j} a_i^2 a_j^2 \leq \| \mathbf{a} \|^4 = 1,$$

and if the random vector \mathbf{B} is of Type 3, then

$$E(z^4) = \sum_{i=1}^D 3a_i^4 + \sum_{i \neq j} a_i^2 a_j^2 \leq 3\| \mathbf{a} \|^4 = 3.$$

Equation (7.24) is proved. \square

We now prove Theorem 7.2.

Proof of Theorem 7.2. Similar to the proof for Theorem 7.1, we set

$$\mathbf{a} = \frac{\mathbf{u} - \mathbf{v}}{\| \mathbf{u} - \mathbf{v} \|}, \quad \mathbf{z} = f(\mathbf{a}) \quad \mathbf{y} = \sqrt{k}\mathbf{z}.$$

Then, by Lemma 7.6 and following the second part of the proof for Theorem 7.1, we obtain

$$\Pr(\| \mathbf{y} \|^2 \geq (1 + \varepsilon)k) < \exp\left(-\frac{k(\varepsilon^2/2 - \varepsilon^3/3)}{2}\right).$$

To evaluate $\Pr(\|\mathbf{y}\|^2 \leq (1 - \varepsilon)k)$, we employ the inequality

$$\mathbb{E}(\exp(-hy_i^2)) \leq 1 - h\mathbb{E}(y_i^2) + \frac{h^2}{2}\mathbb{E}(y_i^4) \leq 1 - h + \frac{3h^2}{2},$$

which yields

$$\begin{aligned} \Pr(\|\mathbf{y}\|^2 \leq (1 - \varepsilon)k) &< \mathbb{E}(\exp(-h\|\mathbf{y}\|^2))^k \exp(hk(1 - \varepsilon)) \\ &\leq \left(1 - h + \frac{3h^2}{2}\right)^k \exp(hk(1 - \varepsilon)). \end{aligned}$$

Taking $h = \frac{\varepsilon}{2(1 + \varepsilon)}$, we have

$$\begin{aligned} \Pr(\|\mathbf{y}\|^2 \leq (1 - \varepsilon)k) &< \left(1 - \frac{\varepsilon}{2(1 + \varepsilon)} + \frac{3\varepsilon^2}{8(1 + \varepsilon)^2}\right)^k \exp\left(\frac{k\varepsilon(1 - \varepsilon)}{2(1 + \varepsilon)}\right) \\ &\leq \exp\left(-\frac{k(\varepsilon^2/2 - \varepsilon^3/3)}{2}\right). \end{aligned}$$

Thus, we obtain

$$\Pr\left(\frac{\|f(\mathbf{u}) - f(\mathbf{v})\|^2}{\|(\mathbf{u} - \mathbf{v})\|^2} \notin [1 - \varepsilon, 1 + \varepsilon]\right) < \frac{2}{n^2} \leq \frac{2}{n^{2+\beta}},$$

for a $\beta > 0$ when

$$k \geq \frac{(4 + 2\beta)\log n}{\varepsilon^2/2 - \varepsilon^3/3}.$$

The theorem is proved. \square

7.4 Applications of Random Projections

In this section, we introduce some applications of random projection (RP) methods. Random projection is a general data reduction method, in which the original high-dimensional data is simply projected into a low-dimensional subspace without using eigen-decomposition of matrices, it runs extremely fast and stable. Besides, in contrast to other linear methods, such as PCA, RP does not use any optimization criteria. Therefore, it is data-independent. Moreover, it is a computationally simple and efficient method that approximately preserves pairwise distances between the points of a set without introducing significant distortion. Hence it is widely used in many DR applications that need to preserve the local structure of the data and require instantaneous results.

7.4.1 Face Recognition Experiments with Random Projection

In [9], the authors evaluated RP for face recognition under various conditions and assumptions. They compared the performance of RP to PCA. The results indicate that RP can be compared quite favorably with PCA while PR is simpler, more computationally efficient, and data-independent. To apply the random projection method in face recognition, a $D \times d$ random matrix \mathbf{R} is created and the rows of \mathbf{R} are orthogonalized so that they form an orthogonal system in \mathbb{R}^D . The main reason to use the orthogonalization is to better preserve the similarities between the original vectors. If the dimension of the original data is high enough, due to the fact that in high-dimensional spaces, there exists a much larger number of almost orthogonal vectors, then the rows of the created random matrix are nearly orthogonal (see Section 1.3.). Hence, the orthogonalization step can be skipped.

Since RP method is data-independent, it does not require to update the projection when the new faces are added to the database or old faces are deleted from it. By contrast, after the database is changed, most deterministic dimensionality reduction techniques require recomputing the eigenfaces and re-projecting the faces to maintain their optimality. As we mentioned in Section 7.1, RP has much lower computational expense comparing to PCA. Performing an RP with an $n \times m$ random matrix of Type 1 needs $O(n \times m^2)$ time of operations, and with an $n \times m$ random matrix of Type 2 or 3 needs $O(n \times m)$, while PCA needs $O(n^2 \times m)$. When $m \ll n$, RD costs only $O(n)$ while PCA costs $O(n^2)$.

RP has several other properties that benefit face recognition. For example, when the original data form highly eccentric Gaussian clusters, RP preserves the clustering effect down in lower dimensions and produces more spherical clusters [10], which could improve recognition performance. In addition, it has been shown that for moderate dimensions, the inner product between the mapped vectors follows closely the inner product of the original vectors, that is, random mappings do not introduce distortion in the data and preserve similarities approximately.

The RP algorithm for face recognition is very similar to the eigenface method introduced in Section 5.3, except that the projection matrix in RP is computed randomly. Assume that the resolution of the face images in the database \mathcal{X} is $D = r \times c$, where $r \times c$ is the image dimension, and there are n faces in the database, $|\mathcal{X}| = n$. Assume also that the faces are already formatted as a $D \times n$ matrix \mathbf{X} , where each column represents a face. Then the RP face projection algorithm consists of the following steps.

- **Compute the average face.** This simple step produces an average face $\bar{\mathbf{x}}$ of all faces in \mathcal{X} .
- **Centralize the faces in the database.** This step produces a centered data. $\hat{\mathbf{x}}_i = \mathbf{x}_i - \bar{\mathbf{x}}$. We denote the new data matrix by $\hat{\mathbf{X}}$.

- **Generate the random matrix.** This step generates an $m \times n$ random matrix \mathbf{R} of Types 1, 2, or 3. Usually, $m \ll n$. If n is very large, then normalize \mathbf{R} to $\mathbf{P} = \frac{1}{m}\mathbf{R}$. Otherwise, if n is moderate, orthogonalize its rows to produce a matrix $\mathbf{P} \in \mathfrak{O}_{m,n}$.
- **Project the centered faces into the random subspace.** In this step, a reduced set of faces is computed by $\mathbf{Y} = \hat{\mathbf{X}}\mathbf{P}'$, in which each column of \mathbf{Y} is the random projected face.

Some experiments and results of RP for face recognition can be found in [9], where the authors used three popular face databases in their study: ORL in (<http://www.face-rec.org/databases/>), CVL in P. Peer's technical report (<http://www.lrv.fri.uni-lj.si/facedb.html>), and AR in ATT Laboratories Cambridge [11]. In each experiment, the database under experimentation is divided into three subsets: training, gallery, and test. The training set was used only by PCA to compute the eigenspace for the comparison purpose. The detailed reports of the experiments and results are referred to [9].

7.4.2 RP Applications to Image and Text Data

The authors of [9] also applied RP method to the dimensionality reduction of high-dimensional image and text data. In their experiments, the image data were chosen from the monochrome images of natural sciences, in which each image is presented as a matrix of pixel brightness, and in the text data each document is presented as a vector whose i th element indicates (some function of) the frequency of the i th vocabulary term in a given dictionary (see Section 1.2). Document data are often highly sparse or peaked: only some terms from the vocabulary are presented in one document, and most entries of the document vector are zero. Also, document data has a nonsymmetric, positively skewed distribution. In the experiments, random projection method is compared to other well-known dimensionality reduction methods, including some nonlinear ones. The experimental results show that despite the computational simplicity, random projection does not introduce a significant distortion in the DR processing, working well in these two very different application areas. They also test RP method on images corrupted by noise. The results confirm the insensitivity of random projection with respect to impulse noise. Thus, random projection proves a promising alternative to some existing methods in noise reduction. The similar work can be also found in [12–15].

References

- [1] Johnson, W.B., Lindenstrauss, J.: Extensions of Lipschitz mappings into a Hilbert space. *AMS Contemp. Math.* 26, 189–206 (1984).
- [2] Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proc. 30th Symp. on Theory of Computing, vol. 13, pp. 604–613 (1998).
- [3] Achlioptas, D.: Database-friendly random projections. Proc. 20th PODS pp. 274–281 (2001).
- [4] Arriaga, R.I., Vempala, S.: An algorithmic theory of learning: Robust concepts and random projection. *Machine Learning* 63(2), 161–182 (2006). URL <http://dx.doi.org/10.1007/s10994-006-6265-7>. Accessed 1 December 2011.
- [5] Klivans, A.R., Servedio, R.A.: Learning intersections of halfspaces with a margin. Proc. 17th COLT 100, 348–362 (2004).
- [6] Szlam, A.: Non-stationary analysis on datasets and applications. Ph.D. thesis, Yale University (2006).
- [7] Frankl, P., Maehara, H.: The Johnson-Lindenstrauss lemma and the sphericity of some graphs. *Journal of Combinatorial Theory* 44, 355–362 (1988).
- [8] Dasgupta, S., Gupta, A.: An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures and Algorithms* 22(1), 60–65 (1999).
- [9] Goel, N., Bebis, G., Nefian, A.: Face recognition experiments with random projection (2004).
- [10] Dasgupta, S.: Experiments with random projection. Proc. Uncertainty in Artificial Intelligence (2000).
- [11] Martinez, A.M., Benavente, R.: Cvc technical report 24. Ph.D. Thesis, University of Cambridge and ATT Laboratories Cambridge (1998).
- [12] Bingham, E., Mannila, H.: Random projection in dimensionality reduction: applications to image and text data. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '01, pp. 245–250. ACM, New York, NY, USA (2001).
- [13] Kaski, S.: Dimensionality reduction by random mapping: fast similarity computation for clustering. In: Proc. Int. Joint Conf. on Neural Networks, vol. 1, pp. 413–418 (1998).
- [14] Lin, J., Gunopulos, D.: Dimensionality reduction by random projection and latent semantic indexing. In: Proc. of SDM (2003).
- [15] Sulić, V., J.Perš, Kristan, M., Kovačić, S.: Dimensionality reduction for distributed vision systems using random projection. In: International Conference on Pattern Recognition (2010).
- [16] Berry, M.W., Pulatova, S.A., Stewart, G.W.: Algorithm 844: computing sparse reduced-rank approximations to sparse matrices. *ACM Trans Math Softw* 31(2), 252–269 (2005).
- [17] Chan, T.F., Hansen, P.C.: Some applications of the rank revealing QR factorization. *SIAM J. on Sci. Statist. Comput.* 13, 727–741 (1992).
- [18] Chen, Z., Dongarra, J.J.: Condition numbers of Gaussian random matrices. *SIAM J. on Matrix Anal. Appl.* 27, 603–620 (2005).
- [19] Goldstine, H.H., von Neumann, J.: Numerical inverting of matrices of high order II. *Amer. Math. Soc. Proc.* 2, 188–202 (1951).
- [20] Goreinov, S.A., Tyrtyshnikov, E.E., Zamarashkin, N.L.: A theory of pseudo-skeleton approximations. *Linear Algebra and Its Applications* 261, 1–21 (1997).

- [21] Gu, M., Eisenstat, S.C.: Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM J. on Sci. Comput.* 17, 848–869 (1996).
- [22] Martinsson, P.G., Rokhlin, V., Tygert, M.: A randomized algorithm for the approximation of matrices. Tech. Rep. 1361, Dept. of Computer Science, Yale University (2006).
- [23] Tyrtyshnikov, E.: Matrix bruhat decompositions with a remark on the QR (GR) algorithm. *Linear Algebra Appl.* 250, 61–68 (1997).
- [24] Tyrtyshnikov, E., Zamarashkin, N.: Thin structure of eigenvalue clusters for non-Hermitian Toeplitz matrices. *Linear Algebra Appl.* 292, 297–310 (1999).
- [25] Woolfe, F., Liberty, E., Rokhlin, V., Tygert, M.: A fast randomized algorithm for the approximation of matrices. *Appl. Comput. Harmon. Anal.* 25(3), 335–366 (2008).
- [26] Zamarashkin, N., Tyrtyshnikov, E.: Eigenvalue estimates for Hankel matrices. *Sbornik: Mathematics* 192, 59–72 (2001).

Part III

Nonlinear Dimensionality Reduction

Chapter 8 Isomaps

Abstract When a data set resides on a d -dimensional manifold that is not a subspace, linear methods cannot effectively reduce the data dimension. Non-linear methods need to be applied in this case. In this chapter, we introduce Isomap method, which unfolds a manifold by keeping the geodesic metric on the original data set. The description of the method is given in Section 8.1, and the Isomap algorithm is presented in Section 8.2. In Section 8.3, we introduce Dijkstra's algorithm that effectively computes graph distances. The experiments and applications of Isomaps are included in Section 8.4. We present the justification of Isomaps in Section 8.5.

8.1 Isomap Embeddings

8.1.1 Description of Isomaps

In the previous part we introduced the linear DR methods, which work effectively if the points of the observed data concentrate around a hyperplane. However, if the points concentrate around a nonlinear manifold, the linear DR methods do not work well because the geometry of the data is now characterized by its underlying manifold, not a subspace. In this case, the dissimilarities between the points of these data have to be measured by a non-Euclidean metric, for instance, the geodesic metric on the underlying manifold. The following example illustrates the necessity of non-Euclidean metric. Let $\varepsilon > 0$ be very small. Let $\mathbf{x} = [\cos \varepsilon, \sin \varepsilon]^T$ and $\mathbf{q} = [\cos(2\pi - \varepsilon), \sin(2\pi - \varepsilon)]^T$ be two points on observed data, inheriting the geometric structure from the circular arc $\Gamma \subset \mathbb{R}^2$:

$$\Gamma = \left\{ [x_1, x_2]^T \in \mathbb{R}^2 : \quad \begin{cases} x_1 = \cos t, \\ x_2 = \sin t, \end{cases} \quad \varepsilon \leq t \leq (2\pi - \varepsilon) \right\}.$$

Then the Euclidean distance between \mathbf{p} and \mathbf{q} is $2\sin \varepsilon \simeq 2\varepsilon$, while the geodesic distance is near $2(\pi - \varepsilon)$. To describe the similarities of points on Γ , we have to use the geodesic metric, not the Euclidean one.

In general, nonlinear DR methods measure the dissimilarity of points with the aid of neighborhood structure on the data. If two points are not in the same neighborhood, they are considered dissimilar. Then a metric will be defined on the data to measure the dissimilarities described by the neighborhood stricture. The defined metric now does not preserve the Euclidean metric in global, but only (approximately) preserves the Euclidean metric in a neighborhood. The local similarity preservation meets the requirement of many data processes very well. Different nonlinear DR methods adopt different metrics for measuring dissimilarity/similarity. In Isomaps, the dissimilarity is measured by the geodesic metric of the underlying manifold. In practice, since the observed data are finite sets, to measure the true geodesic distances between the data points is infeasible. Hence, technically we use the graph distance to approximate the geodesic one.

We call a mapping an isometric mapping if it preserves the distance defined on the data set. Isomap DR method is a distance preserving method such that the Euclidean metric on the new data is equal to the geodesic metric on the original data. Let f be an isometric mapping from the manifold to a low-dimensional Euclidean space (which is often called the *coordinate space* of the manifold). Then the mapping f realizes the dimensionality reduction for any data set on the manifold. The name of the method, *Isomap*, is an abbreviation of *isometric mapping*. Isomap method was first introduced by Tenenbaum, de Silva, and Langford [1, 2], and now is widely used in many applications. Recent development and the applications of Isomaps, particularly, the application in HSI data analysis, can be found in [3–6] and their references.

Mathematically, Isomap method is based on the following consideration. Assume that an observed data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ lies on a d -dimensional Riemannian manifold $M \subset \mathbb{R}^D$, where the dimension $d \ll D$. We denote the geodesic distance on M by d_M (see Chapter 2). Then there exists a coordinate mapping $f : M \rightarrow \mathbb{R}^d$, $f(\mathbf{x}_i) = \mathbf{y}_i$, which preserves the geodesic distance:

$$d_2(f(\mathbf{x}), f(\mathbf{z})) = d_M(\mathbf{x}, \mathbf{z}), \quad \forall \mathbf{x}, \mathbf{z} \in M.$$

Particularly, we have

$$d_2(\mathbf{y}_i, \mathbf{y}_j) = d_M(\mathbf{x}_i, \mathbf{x}_j), \quad \forall 1 \leq i, j \leq n. \quad (8.1)$$

Since the similarity on the data set $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\}$ is the same as on the data set \mathcal{X} , the data \mathcal{Y} is a DR of \mathcal{X} . However, because the manifold M is underlying, we cannot explicitly construct the isometric mapping f for the DR. Instead, we construct a (approximative) geodesic metric d_M on the data \mathcal{X} . By (8.1), the metric d_M induces a Euclidean metric on \mathcal{Y} . Using the technique introduced in Chapter 7, we can find \mathcal{Y} from the metric.

8.1.2 Geodesic Metric on Discrete Set

The key step of Isomap method is the construction of the geodesic metric on the observed data. As we mentioned above, in practice, we cannot measure the geodesic distances between points of a manifold without the formulation of the manifold. Hence, the Isomap method uses the graph distance to substitute the geodesic distance.

Assume that a neighborhood system is well defined on a data set \mathcal{X} , which creates a graph $G = [\mathcal{X}, E]$ such that $(\mathbf{x}_i, \mathbf{x}_j) \in E$ if and only if \mathbf{x}_i and \mathbf{x}_j are adjacent. To construct the graph metric d_G on G , we define the graph distance between $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ as follows.

1. If $(\mathbf{x}, \mathbf{y}) \in E$, then $d_G(\mathbf{x}, \mathbf{y}) = d_2(\mathbf{x}, \mathbf{y})$.
2. If $\mathbf{x}, \mathbf{y} \notin E$, for a path

$$\gamma = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{s+1})$$

that connects \mathbf{x} and \mathbf{y} with $\mathbf{x} = \mathbf{x}_0$ and $\mathbf{x}_{s+1} = \mathbf{y}$, we define the *path distance* by

$$d_\gamma = d_2(\mathbf{x}_0, \mathbf{x}_1) + \dots + d_2(\mathbf{x}_s, \mathbf{x}_{s+1}).$$

Let Γ be the set of all paths that connect \mathbf{x} and \mathbf{y} . Then the graph distance between \mathbf{x} and \mathbf{y} is defined by

$$d_G(\mathbf{x}, \mathbf{y}) = \min_{\gamma \in \Gamma} d_\gamma(\mathbf{x}, \mathbf{y}).$$

When the data points are dense enough on its domain on M , the graph distance d_G approximates the geodesic distance d_M under certain conditions. To justify the validity of the approximation needs more mathematical deductions, which will be put in Section 8.5.

Computing all graph distances between the points of a data set is computationally expensive, particularly, when the size of the data set is very large. The algorithm developed by Dijkstra in 1959 [7] can be applied to the fast computation. We shall discuss the details of Dijkstra's algorithm in Section 8.3.

8.1.3 Isomap Kernel and its Constant Shift

Assume that the data set \mathcal{X} lies on a manifold $M \subset \mathbb{R}^D$, and f is an isometric map from M to \mathbb{R}^d such that $f(\mathbf{x}_i) = \mathbf{y}_i$. The Isomap kernel for a given graph $G = [\mathcal{X}, E]$ is constructed from the graph metric $\mathbf{D}_G = [d_G(i, j)]$ on G , where the (i, j) -entry $d_G(i, j)$ is the graph distance between \mathbf{x}_i and \mathbf{x}_j . It is easy to verify that the graph distance d_G satisfies the distance axioms.

1. $d_G(i, j) \geq 0$, and $d_G(i, j) = 0 \Leftrightarrow i = j$,
2. $d_G(i, j) = d_G(j, i)$,

3. $d_G(i, j) \leq d_G(i, k) + d_G(k, j)$.

Let $d_M(i, j)$ denote the geodesic distance between \mathbf{x}_i and \mathbf{x}_j . It is obvious that when they are adjacent

$$d_G(i, j) \leq d_M(i, j).$$

When the data set \mathcal{X} is dense enough in its domain on M , the graph distance $d_G(i, j)$ is close to the geodesic distance $d_M(i, j)$ so that the graph metric \mathbf{D}_G well approximates the geodesic metric \mathbf{D}_M . As a result, \mathbf{D}_G also approximates the Euclidean metric on the set $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\}$ very well:

$$\mathbf{D}_G \approx \mathbf{D} \stackrel{\text{def}}{=} [d_2(y_i, y_j)].$$

Let $\mathbf{S} = \mathbf{D}_G \boxtimes \mathbf{D}_G = [d_G^2(i, j)]$ and \mathbf{S}^c be its centered matrix: $\mathbf{S}^c = \mathbf{H} \mathbf{S} \mathbf{H}$ (see Section 6.2). By Theorem 6.1 in Section 6.2,

$$\mathbf{G}^c = -\frac{1}{2}\mathbf{S}^c \tag{8.2}$$

is the Gram matrix of a centered data set \mathcal{Y} (i.e., \mathcal{Y} has zero mean). Applying **PCA** to \mathbf{G}^c , we obtain \mathcal{Y} (see Chapter 5). The matrix \mathbf{G}^c in (8.2) is called the *Isomap kernel*.

When the original data set \mathcal{X} is not dense enough in its domain on M , the graph distances of some point pairs may be much smaller than the geodesic distances. Then, the graph metric \mathbf{D}_G cannot approximate the geodesic metric \mathbf{D}_M very well. As a result, the Isomap kernel \mathbf{G}^c in (8.2) may be not positive semi-definite. Therefore, it cannot serve as a Gram matrix of any data set. A simple idea to solve the problem is to add a positive number τ to the graph distance $d_G(i, j), i \neq j$, as a compensation:

$$d_\tau(i, j) = \begin{cases} d_G(i, j) + \tau & i \neq j, \\ 0 & i = j. \end{cases} \tag{8.3}$$

After the modification, the matrix \mathbf{S} becomes

$$\mathbf{S}_\tau = [d_\tau^2(i, j)], \tag{8.4}$$

and \mathbf{G}^c becomes

$$\mathbf{G}_\tau^c = -\frac{1}{2}\mathbf{S}_\tau^c. \tag{8.5}$$

We claim that there exists a $\tau > 0$ that makes \mathbf{G}_τ^c a psd matrix. Then \mathbf{G}_τ^c can be used as a modification of the Isomap kernel. We call \mathbf{G}_τ^c a *constant-shifted Isomap kernel*.

To confirm the existence of such a τ and then give a formula to compute τ , we write $\mathbf{S}_\tau = \mathbf{S} + 2\tau\mathbf{D} + \tau^2(\mathbf{E} - \mathbf{I})$, where \mathbf{E} is the matrix whose entries are 1. Then

$$\mathbf{G}_\tau^c = -\frac{1}{2}(\mathbf{H} \mathbf{S} \mathbf{H} + 2\tau\mathbf{H} \mathbf{D} \mathbf{H} + \tau^2 \mathbf{H} (\mathbf{E} - \mathbf{I}) \mathbf{H}) \tag{8.6}$$

$$= \frac{1}{2}(\tau^2 \mathbf{H} - 2\tau \mathbf{D}^c - \mathbf{S}^c), \quad (8.7)$$

which is a quadratic form of $\tau \geq 0$. When τ is large enough, the matrix \mathbf{G}_τ^c is positive semi-definite and the problem is solved.

To find a lower bound of τ , we study the eigenvalues of \mathbf{G}_τ^c . The vector $\mathbf{1} = [1, 1, \dots, 1]'$ is a trivial eigenvector of \mathbf{G}_τ^c achieving the eigenvalue 0. Then the other eigenvectors are perpendicular to $\mathbf{1}$. Let λ_{\min} be the smallest eigenvalue of $-\mathbf{S}^c$ (which is negative) and μ_{\min} be the smallest eigenvalue of $-\mathbf{D}^c$. Let τ^+ be the positive root of the quadratic equation

$$\tau^2 + 2\tau\mu_{\min} + \lambda_{\min} = 0. \quad (8.8)$$

Solving Equation (8.8), we obtain

$$\tau^+ = \sqrt{\mu_{\min}^2 - \lambda_{\min}} - \mu_{\min}. \quad (8.9)$$

We claim that \mathbf{G}_τ^c is positive semi-definite when $\tau \geq \tau^+$. Indeed, for each centered vector \mathbf{z} , we have

$$\mathbf{z}' \mathbf{G}_\tau^c \mathbf{z} \geq \|\mathbf{z}\|^2 (\tau^2 + 2\tau\mu_{\min} + \lambda_{\min}) \geq 0.$$

Thus, the minimal value of τ that makes \mathbf{G}_τ^c positive semi-definite is no greater than τ^+ in (8.9). This minimal value is given in the following theorem, which was proved by Cailliez in [8].

Theorem 8.1. *The minimal number τ^* that makes \mathbf{G}_τ^c positive semi-definite is the largest eigenvalue of the matrix*

$$\mathbf{B} = \begin{pmatrix} 0 & -\mathbf{S}^c \\ -I & 2\mathbf{D}^c \end{pmatrix}. \quad (8.10)$$

Proof. For a given vector \mathbf{z} , $h(\tau) \stackrel{\text{def}}{=} \mathbf{z}' \mathbf{G}_\tau^c \mathbf{z}$ is a quadratic polynomial of τ . Hence, there is a $\tau(\mathbf{z})$ such that $h(\tau(\mathbf{z})) = 0$ and $h(\tau) > 0$ for all $\tau > \tau(\mathbf{z})$. We agree that $\tau(\mathbf{z}) = -\infty$ if the quadratic form $h(\tau) > 0$ for all τ . It is also clear that $\tau(\mathbf{z}) = \tau(k\mathbf{z})$ for $k \in \mathbb{R} \setminus \{0\}$. Because \mathbf{G}^c is not positive semi-definite, there is at least one \mathbf{z}_0 making $\mathbf{z}_0' \mathbf{G}^c \mathbf{z}_0 < 0$ and $\tau(\mathbf{z}_0) > 0$. On the other hand, $\tau(\mathbf{z}) \leq \tau^+$ for all \mathbf{z} . Let

$$\tau^* = \sup_{\mathbf{x}} \tau(\mathbf{x}). \quad (8.11)$$

Then, $0 < \tau^* \leq \tau^+$. Since $\tau(\mathbf{z})$ is a continuous function of \mathbf{z} , there is a \mathbf{z}^* such that $\tau(\mathbf{z}^*) = \tau^*$. We have

$$\mathbf{z}' \mathbf{G}_\tau^c \mathbf{z} \geq 0, \quad \forall \mathbf{z}, \tau \geq \tau^*, \quad (8.12)$$

$$(\mathbf{z}^*)' \mathbf{G}_{\tau^*}^c \mathbf{z}^* = 0. \quad (8.13)$$

We now prove τ^* is the largest eigenvalue of \mathbf{B} . First, we claim that τ^* is an eigenvalue of \mathbf{B} . Let $\mathbf{w} = [\mathbf{y}, H\mathbf{z}^*]'$, where $\mathbf{y} = -\tau^* \mathbf{S}^c \mathbf{z}^*$. Then, we have

$$-\mathbf{S}^c \mathbf{z}^* = \tau^* \mathbf{y} \quad (8.14)$$

and, by (8.7) and (8.13),

$$(\tau^{*2} \mathbf{H} - 2\tau^* \mathbf{D}^c - \mathbf{S}^c) \mathbf{z}^* = 0. \quad (8.15)$$

As a consequence of (8.14) and (8.15), we obtain

$$\mathbf{y} + \tau^* \mathbf{H} \mathbf{z}^* - 2\mathbf{D}^c \mathbf{H} \mathbf{z}^* = 0. \quad (8.16)$$

Equations (8.14) and (8.16) yield

$$\begin{pmatrix} 0 & -\mathbf{S}^c \\ -\mathbf{I} & 2\mathbf{D}^c \end{pmatrix} \begin{pmatrix} \mathbf{y} \\ \mathbf{H} \mathbf{z}^* \end{pmatrix} = \tau^* \begin{pmatrix} \mathbf{y} \\ \mathbf{H} \mathbf{z}^* \end{pmatrix},$$

which proves that τ^* is an eigenvalue of \mathbf{B} . In order to prove τ^* is the largest eigenvalue of \mathbf{B} , we assume that σ is an arbitrary eigenvalue of \mathbf{B} associated with the eigenvector $[\mathbf{v}, \mathbf{u}]'$:

$$-\mathbf{S}^c \mathbf{u} = \sigma \mathbf{v}, \quad (8.17)$$

$$-\mathbf{v} + 2\mathbf{D}^c \mathbf{u} = \sigma \mathbf{u}. \quad (8.18)$$

By (8.17), \mathbf{v} is a centered vector since it is in the range of $-\mathbf{S}^c$. In (8.18), since both $2\mathbf{D}^c \mathbf{u}$ and \mathbf{v} are centered vectors, so is \mathbf{u} . Thus, $\mathbf{u} = \mathbf{H} \mathbf{u}$, and we have

$$(-\mathbf{S}^c - 2\sigma \mathbf{D}^c + \sigma^2 \mathbf{H}) \mathbf{u} = 0,$$

which yields $\mathbf{u}' \mathbf{G}_\sigma^c \mathbf{u} = 0$, i.e., $\tau(\mathbf{u}) = \sigma$. By (8.11), $\sigma \leq \tau^*$. The proof is completed. \square

The technique used in (8.3) is called the *constant-shift technique*, which is not only used for the modification of Isomap kernel, but also used whenever a symmetric matrix needs to be altered into a positive semi-definite one.

8.2 Isomap Algorithm

8.2.1 Algorithm Description

The Isomap algorithm adopts the strategy described in Diagram 1 (in Section 4.2): (1) Compute the graph metric on the data set \mathcal{X} that approximates the geodesic metric of the underlying manifold, and (2) apply a linear method, such as PCA, to find the low-dimensional data set that preserves the graph metric. The Isomap algorithm consists of four steps.

Step 1. Neighborhood definition. The neighborhood of a point in the set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ can be constructed by either its k -neighborhood or ε -neighborhood. If the notion of k -neighborhood is used, then the method is called a k -Isomap, or else, it is called an ε -Isomap. We denote the neighborhood of \mathbf{x}_i by $O(i)$.

Step 2. Graph distance computation. The defined neighborhood system creates a graph $G = [\mathcal{X}, E]$ on the data set. We compute the graph metric on \mathcal{X} in terms of the graph distance $d_G(i, j)$ between \mathbf{x}_i and \mathbf{x}_j for each pair of points on \mathcal{X} . Recall that if k -neighborhood is adopted, then $d_G(i, j)$ may not be equal to $d_G(j, i)$ since $\mathbf{x}_j \in O_i \not\Rightarrow \mathbf{x}_i \in O_j$. To ensure $d_G(j, i) = d_G(i, j)$, we may either define the graph distance between \mathbf{x}_i and \mathbf{x}_j by $\min(d_G(i, j), d_G(j, i))$, or modify the digraph G to the undirected graph. If there is no path connecting \mathbf{x}_i and \mathbf{x}_j , then we set $d_G(i, j) = \infty$.

Step 3. DR kernel construction. Assume the points of \mathcal{X} is dense enough. Then \mathbf{D}_G is Euclidean. Compute the matrices $\mathbf{S}_G = [d_G^2(i, j)]$ and

$$\mathbf{G}^c = -\frac{1}{2}\mathbf{H}\mathbf{S}_G\mathbf{H},$$

where \mathbf{G}^c is the kernel of Isomap DR method. Otherwise, to ensure that the kernel is positive semi-definite, the constant-shift technique may be applied (see Remark 8.2 below).

Step 4. Eigen decomposition. Let the eigen decomposition of \mathbf{G}^c be given by

$$\mathbf{G}^c = \mathbf{V}\Lambda\mathbf{V}',$$

where $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_d, \dots, \lambda_n)$ and $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n]$. Let $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_n]$ be determined by

$$\mathbf{Y}' = [\sqrt{\lambda_1}\mathbf{v}_1, \sqrt{\lambda_2}\mathbf{v}_2, \dots, \sqrt{\lambda_d}\mathbf{v}_d].$$

Then \mathbf{Y} is the dimensionality reduction of \mathbf{X} . By Theorem 6.3, we have the following error estimate.

$$\frac{1}{2n} \sum_{i,j=1}^n (d_G^2(\mathbf{x}_i, \mathbf{x}_j) - d_2^2(\mathbf{y}_i, \mathbf{y}_j)) \leq \sum_{l=d+1}^n \lambda_l.$$

Remark 8.1. To compute the graph distance between two points, the path distances of all paths that connect them must be estimated. Floyd's algorithm [9] employs an exhaustive iteration method computing all possible path distances to achieve the minimum. However, this is very time-consuming. Floyd's algorithm requires $O(n^3)$ computations. For large n , Dijkstra's algorithm [7] reduces the computation to $O(n^2 \log n)$. We shall discuss Dijkstra's algorithm in the next section.

Remark 8.2. As discussed in Section 8.1, the matrix \mathbf{D}_G may not approximate \mathbf{D}_M well if the points of the data set are not dense enough in the domain. In this case, \mathbf{G}^c may not be positive semi-definite. To ensure that \mathbf{G}^c is psd, we have to apply the constant-shift technique to modify \mathbf{G}^c . Thus, Step 3 above is split into the following two steps.

Step 3-1. Computation of shifting constant. Centralize \mathbf{D}_G and \mathbf{S}_G by the formulas

$$\mathbf{F}^c = -\frac{1}{2}\mathbf{H}\mathbf{D}_G\mathbf{H},$$

$$\mathbf{G}^c = -\frac{1}{2}\mathbf{H}\mathbf{S}_G\mathbf{H},$$

and compute the largest eigenvalue t of the matrix

$$\begin{pmatrix} 0 & 2\mathbf{G}^c \\ -I & -4\mathbf{F}^c \end{pmatrix}.$$

Step 3-2. Constant-shift adjustment. Compute

$$\mathbf{G}^c = \mathbf{G}^c + 2t\mathbf{F}^c + \frac{1}{2}t^2\mathbf{H},$$

which is used to replace the kernel generated in Step 3.

8.2.2 Matlab Code of Isomap

The first Isomap algorithm was developed by Tenenbaum, de Silva, and Langford [1, 2]. The presented Isomap DR code is a modification of the original. The algorithm consists of four parts: data-graph construction, graph distance computation, Isomap kernel generation, and spectral decomposition of the kernel. The input data of the algorithm is a data matrix \mathbf{X} , in which each column is a data vector.

1. The first part of the algorithm is common for all nonlinear dimensionality reduction methods. An option is set for choosing either k -neighborhood or ε -neighborhood for the graph construction. Usually, k -neighborhood is more common than ε -neighborhood. When k -neighborhood is used, we also adjust it to generate an undirected graph.
2. The second part of the algorithm is the computation of graph metric. In the presented code, Dijkstra's algorithm is applied to the computation. The code adopts the .dll file `perform_dijkstra_fast.dll` for a fast performance. If the code is not run on a Windows platform, the source code `dijkstra.cpp` can be used to build an executive file for the platform where the Matlab is installed. An m-function `syndijkstra.m` is also provided in case that the fast executive file does not exist. But this Matlab code is several orders of magnitude slower than the C-based mex-file.
3. In the third part of the code, Isomap kernel is generated by the formula (8.2).
4. The last part makes the spectral decomposition of the kernel to produce the DR data set.

Remark 8.3. The constant-shift adjustment is not included in this code. Readers can develop it by themselves, following the instruction in the previous section or consulting the paper [10]. When the dimension of the Isomap kernel is large, the EMPCA algorithm can be applied in the fourth step to accelerate the computing speed in the eigen decomposition (see Subsection 5.2.2).

The following is the Matlab code of Isomap dimensionality reduction.

```

function Y = drisomap(X,ndims,options)
% Isomap DR
% SYNTAX:
%   Y = Isomap(X,ndims,options);
% INPUTS:
%   'X': a D x N matrix
%           D = dimension, N = #points
%   'ndims': Dimension of output data set.
% OPTIONS:
%   'options.epsilon' : construct data graph
%           using epsilon-neighbors.
%   'options.k' : using k-neighbors.
%   'options.verb': display the processing verbally.
%
% CALL: data_neighborhood
%           dijkstra_fast.dll or dijkstra
% Reference:
% Tenenbaum, de Silva, and Langford,
% Isomap code -- (c)1998-2000, Josh Tenenbaum
%
% Initialize options.
options.symmetric = 1;
if isfield(options, 'verb')
    verb =options.verb;
else
    verb =1;
end
% Part 1: Construct data-graph.
if verb
    disp('- construct data graph');
end
D2 = data_neighborhood(X,options);

% Part 2: Compute graph distance.
if verb
    disp('- computing graph distance matrix');
end
N = size(D2,1);
D1 = sqrt(D2);
% Try to use .dll for faster performance.
if exist('dijkstra_fast.dll','file')
```

```

DG = fast_dijkstra_fast(D1, 1:N);
else
    DG = symdijkstra(D1, 1:N);
end
% Part 3: Generate Isomap kernel
if verb
    disp('- Generate Isomap kernel');
end
DG = DG.^2;
GC = -.5*(DG - sum(DG)'*ones(1,N)/N ...
    - ones(N,1)*sum(DG)/N + sum(DG(:))/(N^2));
% Part 4: Kernel decomposition.
if verb
    disp('- Output low-dimensional data');
end
opt.disp = 0; opt.isreal = 1; opt.issym = 1;
[Y, val] = eigs(GC, ndims, 'LR', opt);
for i=1:ndims
    Y(:,i) = Y(:,i)*sqrt(val(i,i));
end

```

8.3 Dijkstra's Algorithm

8.3.1 Description of Dijkstra's Algorithm

In Isomap algorithm, the computation of the graph distances between points in data set costs the most time. Assume that a graph has been constructed. To compute the distance between two nodes that are not directly connected by an edge, we need to compute all possible path distances and then minimize the results. An exhaustive iterative method known as Floyd's algorithm [9] is optimal for dense graphs (that is, the adjacency matrices are dense). Floyd's algorithm costs $O(n^3)$. However, most data graphs for dimensionality reduction are constructed by neighborhood systems, either k -neighborhood or ε -neighborhood. Therefore, their adjacency matrices are sparse. For sparse graphs, the computing cost can be reduced by using Dijkstra's algorithm [7], conceived by Dijkstra in 1959. The algorithm solves the shortest path searching problem for a weighted graph, producing a shortest path tree. Using the path tree, the graph distances computed in advance are reusable for computing the graph distances of new pairs.

The idea of Dijkstra's algorithm is simple. Suppose that you want to find the shortest path between two intersections on a geographic map, from a starting point to a destination. To accomplish this, you could highlight

the streets (tracing the streets with a marker) in a certain order until you have a route highlighted from the starting point to the destination. The order is conceptually simple: at each iteration, create a set of intersections consisting of every unmarked intersection that is directly connected to a marked intersection, this will be your set of considered intersections. From that set, find the closest intersection to the destination and highlight it and mark the street to that intersection, draw an arrow with the direction, then repeat. In each stage mark just one new intersection. When you reach the destination, the drawn arrows indicate the shortest path.

We demonstrate the steps of Dijkstra's algorithm in Fig. 8.1. For convenience, we call the node at which we are starting the initial node, and call the distance from the initial node to a node, say b , the distance of b . At the initial step, Dijkstra's algorithm assigns some initial distance values for all nodes, and then improves them step by step. In our demonstration, the first figure is a subgraph consisting of 6 nodes. The initial node is Node 1, marked as a , and the destination is Node 6, marked as b . The boxed number on each edge is the edge distance. We shall calculate all graph distances from a (Node 1) to other nodes.

Step 1. This is the initial step. We assign to every node a distance value.

Since we assume that the initial node is a , we set the distance of Node 1 at zero. The distances of all other nodes are set at infinity. In the figures, the distance of a node is shown nearby. At the initial step, all nodes are considered unvisited, except the initial node a that is set as current.

Step 2. Calculate all tentative distances from the initial node to its neighbors. There are three nodes, Nodes 2, 3, and 6, directly connected to Node 1 by edges. The distance of Node 1 is 0, and the edge distance from Node 1 to Node 2 is 1. Hence we get the tentative distance $0 + 1 = 1$ for Node 2. Similarly, we calculate the tentative distances 8 and 14 of Node 3 and Node 5, respectively. Mark these tentative distances. Since all distances of the nodes directly connecting Node 1 are calculated, Node 1 will not be used in the further calculation. We mark it as **out**.

Step 3. We set Node 2 as the current node, having the distance 1. It has two visitable neighbors: Nodes 3 and 4. The path distance of the path of Node 1 → Node 2 → Node 3 is calculated by adding the edge distance 3 to the distance of Node 2. We get the distance $1 + 5 = 6$, which is smaller than the current tentative distance 8. Hence, we update the distance of Node 3 to 6. The tentative distance of Node 4 is $10 + 1 = 11$. Record all computed distances, and mark Node 2 as **out**.

Step 4. We now set Node 3 as the current node, which has the distance 6. In the similar way, we calculate the distances of Node 5 and Node 4 starting from Node 3. The results are 12 and 11 respectively. Update the distance of Node 5 to 12 and keep 11 for Node 4. We then set Node 3 as **out** and make Node 4 and Node 5 the current nodes.

Step 5. Finally, we calculate the distance of Node 6 starting from Node 5 and Node 4 respectively, getting 16 and 13. We choose 13 as the distance

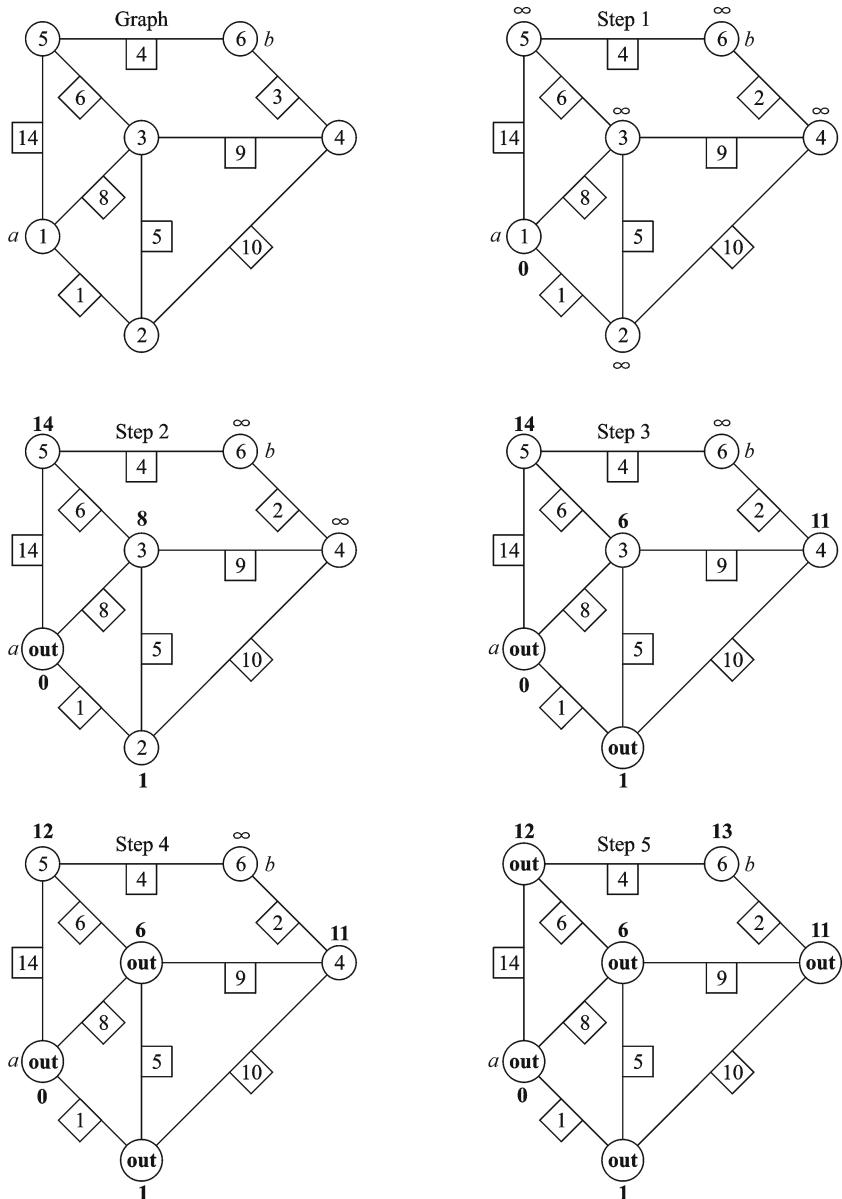


Fig. 8.1 Dijkstra's algorithm runtime. The figure illustrates the Dijkstra's algorithm runtime for a subgraph with 6 nodes. The graph distances from Node 1 to other nodes is computed using the algorithms step by step.

of Node 6. The computation of all distances from Node 1 to other nodes is completed. Particularly, the distance from a to b is 13. The computed distances of the nodes can be used in the further computation, For example, if b (Node 6) has neighbors Nodes 7 and 8, then the path distances from a to Node 7 and to Node 8 can be calculated by adding 13 to the edge distances Node 6 → Node 7 and Node 8, respectively.

8.3.2 Matlab Code of Dijkstra's Algorithm

Dijkstra's algorithm computes the graph metric on a given graph, which is either undirected or directed. Let $G = [P, \mathbf{W}]$ be a given graph, where P is the label set of the nodes and \mathbf{W} is the weight matrix. In Matlab code, the first input of the algorithm is the weighted matrix \mathbf{W} , which can be formatted to either a full matrix or a sparse one. If \mathbf{W} is formatted to a sparse matrix, then it only records the weights on edges. The other two optional inputs are point sets, where the first one is the starting point set S , and the second one is the destination point set T . The distances from a point in S to all points in the set T are computed and stored in a row of the output $|S| \times |T|$ matrix \mathbf{D} . After the distances from a point to all points in T are calculated, the algorithms shift to the next point in the set S and do the similar computation, until all distances are calculated. In the previous subsection, we already learned how to compute the graph distance from a point to another one using Dijkstra's algorithm. The calculation of the distances from one point to all points in a set is similar. For example, let u be a node in the set S . We want to compute the distances from u to all points in the set S , assuming the edge distance from u to itself is 0. Write $|P| = n$ and assume that the label of u is 1. We initialize a distance vector $\mathbf{d} = [0, \text{Inf}, \text{Inf}, \dots, \text{Inf}]$ for u . The first step is assigning the distances of all neighbors of u by their edge distance (stored in \mathbf{W}), storing them in a vector \mathbf{d} , and letting u be out. The second step is choosing the point other than u (i.e., a not-out point) having the minimal distance, say v , as the new start-point, finding all of its neighbors, computing and updating the path distances from u to the neighbors of v , storing them into \mathbf{d} , and then letting v be out. Repeat this step until all points are out. The following code computes the distances from a point u to a set T .

```

function dv = pDijkstra(W, T)
% Compute the distances from a point u to a Set
% INPUTS:
% % W: weight adjacency matrix
% % (non-negative and symmetric)
n = size(W,1);
if nargin<2
    T = 1:n;
end

```

```

if size(T,1)~=1
    T = (T(:))';
end
% Initialize distance vector.
v = Inf*ones(1,n); v(1)=0;
% Initialize the position vector for Node set.
Plabel = 1:n;
% Initialize the mode vector for Node set.
NotOut = true(n,1);
% Transpose weight matrix W for easy
% finding neighbors of nodes.
W = W';
% The main loop
for k=1:n
    % compute n distances from a point to T.
    % Select a node from the points
    %           that are not out yet.
    [vi, i] = min(v(NotOut));
    % Find the label of the node.
    j = Plabel(i);
    % Make the node out.
    Plabel(i) = [];
    % Change the mode of the node.
    NotOut(j) = false;
    % Find all neighbors of the node.
    [nghbr_list, c, nghbr_values] = find(W(:,j));
    % Compute new path distances using edge distances,
    % then update the path distances from u to them.
    if ~isempty(nghbr_list)
        newpathD = vi + nghbr_values;
        v(nghbr_list) = min(v(nghbr_list),newpathD);
    end
end
% Output the distance vector from u to T.
dv = v(T);

```

A more general problem is to compute all graph distances from the points of a source set S to the points of a target set T . Then the output is a $|S| \times |T|$ matrix. The algorithm essentially repeats the algorithm above. We present the code, which is a modification of the original code written by M. G. Kay at North Carolina State University in 2000.

```

function D = dijkstram(A, s, t)
% compute graph distance matrix
%           using Dijkstra algorithm.
% SYNTAX:
%   D = dijkstra(A,plist)
% INPUTS:
%   A: n x n weighted adjacency matrix of graph
%   s: From node indices

```

```

%      [] (default), paths from all nodes
%      t: To node indices
%      [] (default), paths to all nodes
% OUTPUT:
%      D:  =[D(i,j)], |s| x |t| graph distance matrix
%          from 's' to 't', where D(i,j)
%          is distance from node 'i' to node 'j'
%
% Copyright (c) 1998-2000 by Michael G. Kay
% Matlab Version 1.3 29-Aug-2000
%
% Modified by JBT, Dec 2000, to delete paths

% Input Error Checking
error(nargchk(1,3,nargin));

[n,cA] = size(A);
% set s and t to column vectors.
if nargin < 2 || isempty(s)
    s = (1:n)';
else
    s = s(:);
end
if nargin < 3 || isempty(t)
    t = (1:n)';
else
    t = t(:);
end

if ~any(any(tril(A) ~= 0))
    % A is upper triangular
    isAcyclic = 1;
elseif ~any(any(triu(A) ~= 0))
    % A is lower triangular
    isAcyclic = 2;
else
    % Graph may not be acyclic
    isAcyclic = 0;
end

% check the validity of inputs
if n ~= cA
    error('A must be a square matrix');
elseif ~isAcyclic && any(any(A < 0))
    error('A must be non-negative');
elseif any(s < 1 | s > n)
    error(['"s" must be an integer...
        between 1 and ',num2str(n)]);
elseif any(t < 1 | t > n)
    error(['"t" must be an integer...
        between 1 and ',num2str(n)]);
end

```

```

        between 1 and ',num2str(n)]);
end
% End (Input Error Checking)
% Use transpose to speed-up FIND for sparse A
A = A';

D = zeros(length(s),length(t));
P = zeros(length(s),n);

% Compute the graph distance from i to j
for i = 1:length(s)
    j = s(i);
    Di = Inf*ones(n,1); Di(j) = 0;
    isLab = zeros(length(t),1);
    if isAcyclic == 1
        nLab = j - 1;
    elseif isAcyclic == 2
        nLab = n - j;
    else
        nLab = 0;
        UnLab = 1:n;
        isUnLab = true(n,1);
    end

    while nLab < n && ~all(isLab)
        if isAcyclic
            Dj = Di(j);
        else % Node selection
            [Dj,jj] = min(Di(isUnLab));
            j = UnLab(jj);
            UnLab(jj) = [];
            isUnLab(j) = false;
            %isUnLab(j) = 0;
        end
        nLab = nLab + 1;
        if length(t) < n
            isLab = isLab | (j == t);
        end
        [jA,kA,Aj] = find(A(:,j));
        Aj(isnan(Aj)) = 0;
        if isempty(Aj)
            Dk = Inf;
        else
            Dk = Dj + Aj;
        end
        P(i,jA(Dk < Di(jA))) = j;
        Di(jA) = min(Di(jA),Dk);
        % Increment node index for upper triangular A
        if isAcyclic == 1
            j = j + 1;
        end
    end
end

```

```

    % Decrement node index for lower triangular A
    elseif isAcyclic == 2
        j = j - 1;
    end
    %disp( num2str( nLab ) );
end
D(i,:) = Di(t)';

```

The above code can be used to compute graph distances for the weighted graphs of any type. In Isomaps, we only need to compute graph metrics on undirected graphs. Taking the advantage of the symmetry of undirected graph, we develop a graph metric computation algorithm that consists of a forward phase and a backward phase. In the forward phase, each time we select a new node as the starting point, we only compute the path distances from this node to such points that the distances from them to other points have not been computed yet. In this phase, a path tree is made to indicate the computed shortest paths. In the backward phase, we compute the path distances utilizing the path tree developed in the forward phase, avoiding the repeated computation. Therefore, in this phase, only the graph distances that were not computed in the forward phase are made-up. The algorithm saves about a half of computational time compared with the regular Dijkstra's algorithm.

```

function D = symdijkstra(A,pfrm,alld)
% calculate undirected-graph distance.
% It utilities the symmetry for fast computation.
% SYNTAX: D = symdijkstra(A,s,t)
% INPUTS:
%     A: n x n symmetric weighted adjacency matrix
%     PFROM: node indices
%         default []: from all nodes
%     ALLD: 1, to all nodes,
%           : [] OR 0 (default), to the same set
% OUTPUT:
%     D: D(i,j)is distance from 'i' to 'j'
%     if ALLD=0, |s| x |s| symmetric matrix
%     if ALLD=1, |s| x n matrix, with the most
%     left |s| x |s| symmetric sub-matrix, which
%     represents distances between the selected
%     point set. The order of points in matrix
%     is as follows. The selected |s| points
%     come first, followed by other points,
%     keeping their orders in the original set.
% Copy right note:
% Modified from Michael G. Kay's code: Matlog
% Version 1.3 Aug-2000, Copyright (c) 1998-2000.
%
%
```

```

% Input Error Checking
error(nargchk(1,3,nargin));
[n,cA] = size(A);
if n ~= cA
    error('A must be a square matrix');
elseif any(any(A < 0))
    error('A must be non-negative');
end
if nargin < 3
    alld = 0;
    if nargin < 2
        pfrm = (1:n);
    end
end
if size(pfrm,1)~=1
    pfrm = (pfrm(:))';
end
% End (Input Error Checking)
% Make the permutation if the selected points
% are not at 1:m.
m=length(pfrm);
if m<n && any(pfrm~=1:m))
    p=1:n;
    p(pfrm)=[];
    b=[pfrm,p];
    A = A(b,b);
end
% Initialize the output matrix.
if alld
    cD=n;
else
    cD=m;
end
D = zeros(m,cD);
for i = 1:m
    %Computing starts all nodes are unlabeled.
    Di = Inf*ones(n+1-i,1); Di(1) = 0;
    NdLbl = 1:(n+1-i);
    NotOut = true(n+1-i,1);
    for k=1:(n+1-i) % Forward phase:
        % compute all path distances
        % through new points
        % Node selection
        [Dj,jj] = min(Di(NotOut));
        j = NdLbl(jj);
        NdLbl(jj) = [];
        NotOut(j) = false;
        [jA,kA,Aj] = find(A(:,j));
        if ~isempty(Aj)
            Dk = Dj + Aj;
            for l=1:length(Aj)
                Dj(l) = Dk;
            end
        end
    end
end

```

```

        Di(jA) = min(Di(jA),Dk);
    end
end
D(i,i:cD) = (Di(1:cD+1-i))';
%Backward phase: compute path distances
% through the Out-points
if i>1
    D(i,i:cD)= min(D(1:i,i:cD)+ ...
    repmat(D(1:i,i),1,cD-i+1));
end
%update weighted matrix for
%avoiding repeated computing.
A(1,:)=[]; A(:,1)=[];
end
% make the first block symmetric.
D(1:m,1:m)=D(1:m,1:m)+(D(1:m,1:m))';

```

Finally, we remark that the Matlab code for graph metric computation is very slow. Faster computation should be implemented in C or other computer languages.

8.4 Experiments and Applications of Isomaps

8.4.1 Testing Isomap Algorithm on Artificial Surfaces

In Subsection 5.2.3, we demonstrated that linear dimensionality reduction methods do not work well for the data residing on surfaces. We now show that Isomap algorithm works well for surface data. We use 5 artificial surfaces, Square, 3-D cluster, Punched sphere, Swiss roll, and S-curve to test Isomap algorithm. We show the test results for the first three surfaces, Square, 3-D cluster, and Punched sphere, in Fig. 8.2. The graphs of these three surfaces are displayed on the top line, and their DR data on the plane are displayed on the bottom line. It is not surprise that the Square-data are well reduced since the original data is on a hyperplane. Isomap DR algorithm also performs very well for other two surfaces, showing its ability in the dimensionality reduction for the data on uneven surfaces.

In Fig. 8.3, we apply Isomap algorithm to Swiss roll and S-curve using the same settings: 1000 random selected points constitute the data set and 10-neighborhood is used in the construction of the data graph. The experimental results show that Isomap algorithm works very well in the dimensionality reduction of these data.

Punched spheres are non-expandable surfaces. Hence, there are no isometric mappings that can preserve the geodesic distances for them globally.

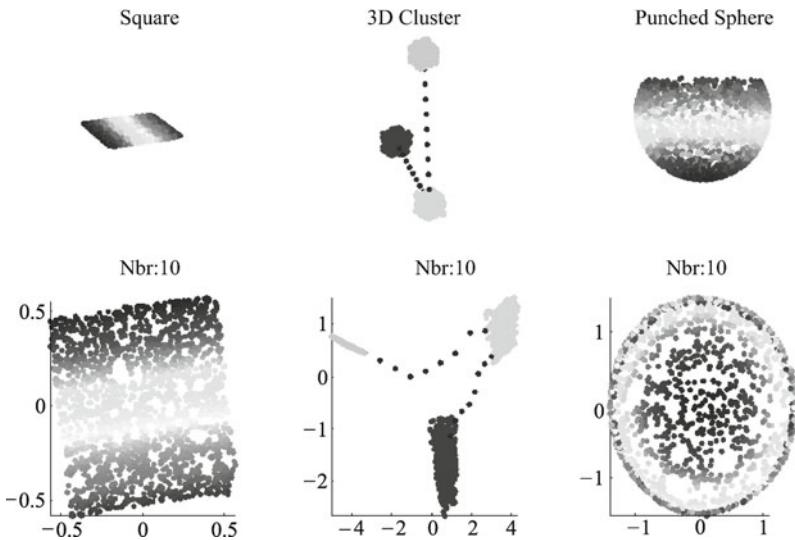


Fig. 8.2 Isomap algorithm is performed on three surfaces. On the top line, the left is a square in space, the middle is a 3D-cluster, and the right is a punched sphere. The data set for each surface contains 1000 random samples. Each data graph is constructed using 10-neighborhood, and the graph distances are computed by Dijkstra's algorithm. Their 2-dimensional DR data sets are displayed on the bottom line correspondingly. It is shown that Isomap algorithm successfully reduces the dimension of nonlinear surface data sets.

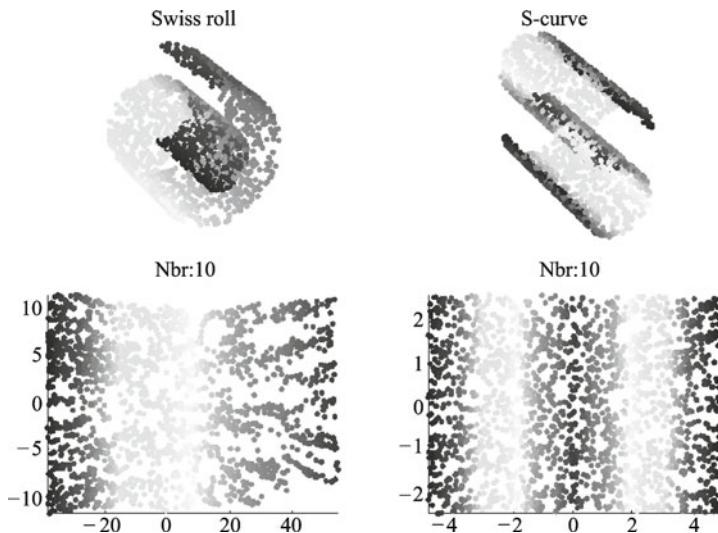


Fig. 8.3 Isomap algorithm is performed on Swiss roll and S-shape. These two are expandable surfaces. The original data are shown on the top line, where the left is Swiss roll and the right is S-curve. Their 2-dimensional DR data are displayed on the bottom line correspondingly. The results show that Isomap algorithm reduces the dimension of these two surface data sets very well.

Figure 8.4 shows that when the height of the punched sphere increases, the separation of the DR data becomes worse. In Fig. 8.4, mixture colors occur in the area near the boundary when the height of the punched sphere becomes great.

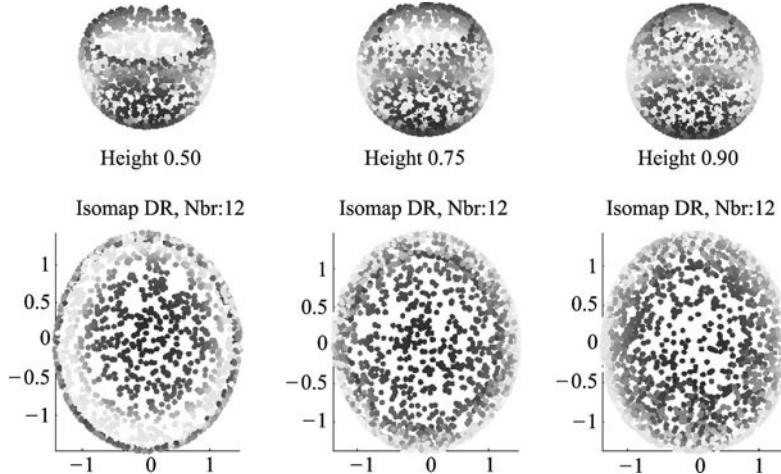


Fig. 8.4 Isomap algorithm is performed on the punched spheres with various heights. On the top line, we present three punched spheres with heights of 0.5, 0.75, and 0.9 respectively. The diameter of the sphere is 1. On the bottom line are their DR data sets correspondingly. The results show that the DR data sets do not preserve the local geometry near the boundary when the punched holes become smaller. It is reasonable that when the heights increase, the graph distances between the points near the punched boundary do not well approximate the geodesic distances, so that large errors occur.

In Fig. 8.5, we display the impact of the neighborhood sizes upon the DR processing. In nonlinear DR methods, choosing large sizes of neighborhoods costs more computation time. However if the neighborhood size is too small, the data graph may not truly reflect the data similarity, causing an incorrect DR processing. The experimental results in Fig. 8.5 show that Isomap algorithm is relatively insensitive to the neighborhood sizes. Note that DR algorithms that are not sensitive to the neighborhood sizes are more desirable. Hence, Isomap method is a promising DR method. Surely, too small neighborhood sizes deform the geometric structure as we discussed in Chapter 4.

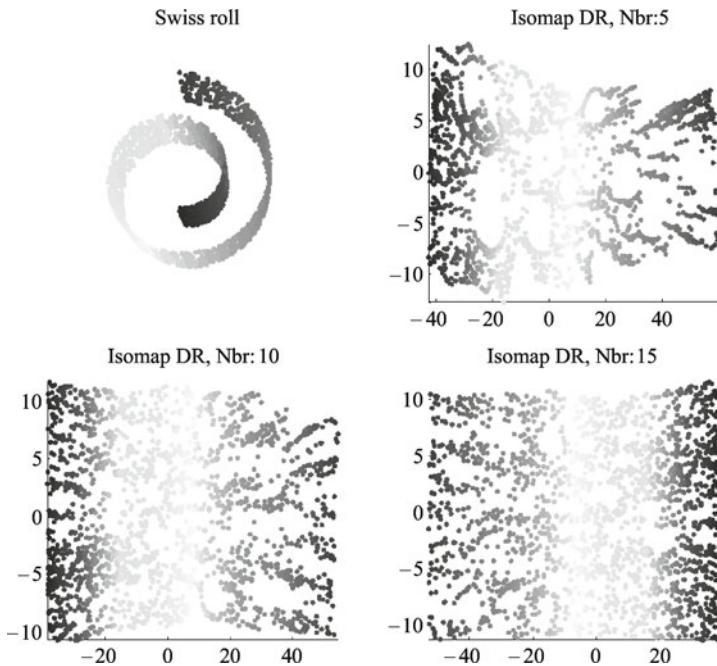


Fig. 8.5 Isomap embeddings of Swiss roll of various sizes of neighborhood. Three different sizes of k -neighborhood, $k = 5, 10, 15$, are used in this experiment. The results show that Isomap algorithm is insensitive to the neighborhood sizes, a wide range of neighborhood sizes can be chosen in the algorithm for producing the similar DR results. However, when the size of neighborhood is too small, say $k = 5$, in this experiment, although the low-dimensional data have a little distortion, the geometry of the original data is still well preserved in most of the area.

8.4.2 Isomap Algorithm in Visual Perception

A canonical problem in dimensionality reduction from the application of visual perception is illustrated in Fig. 8.6 by Tenenbaum, de Silva, and Langford [2]. In the figure, the input consists of many images of a person's face observed under different poses and lighting conditions disorderly. These images are considered as points in a high-dimensional vector space, with each input dimension corresponding to the brightness of one pixel in the image or the firing rate of one retinal ganglion cell. Although the input dimensionality may be quite high (e.g., 4096 for these 64 pixels by 64 pixels images), the perceptually meaningful structure of these images has quite a few independent degrees of freedom. Within the 4096-dimensional input space, all of the images lie on an intrinsically three-dimensional manifold that can be parameterized by

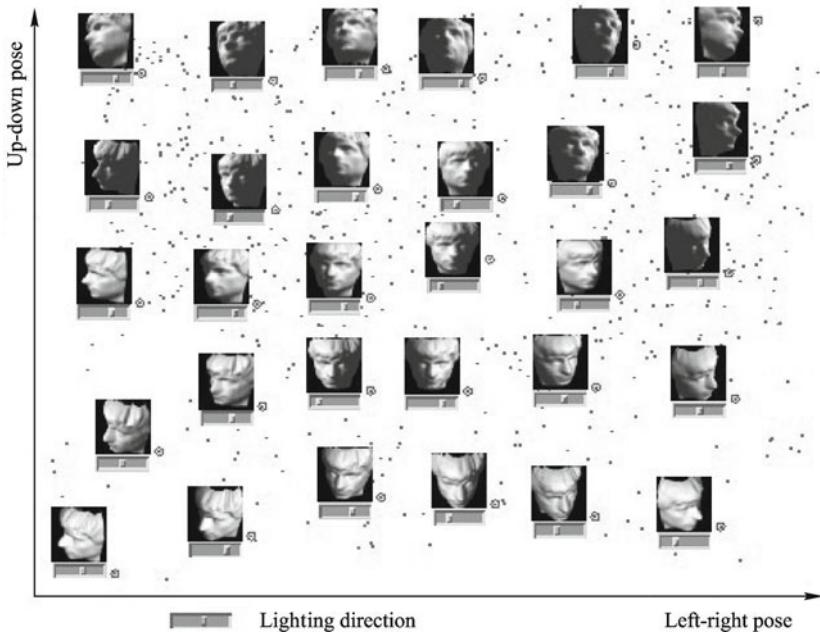


Fig. 8.6 A canonical dimensionality reduction problem from visual perception. The input consists of a sequence of 4096-dimensional vectors, representing the brightness values of 64×64 images of a face to which different poses and lighting directions are rendered. Applied to 698 raw images, Isomap (with 6-neighborhood) learns a three-dimensional embedding of the data's intrinsic geometric structure. A two-dimensional projection is shown, with a sample of the original input images (with circles) superimposed on all the data points and horizontal sliders (under the images) representing the third dimension. Each coordinate axis of the embedding correlates strongly with one degree of freedom underlying the original data: left-right pose (x -axis, $R=0.99$), up-down pose (y -axis, $R=0.90$), and lighting direction (slider position, $R=0.92$). Note: This graph is modified from the original in [2].

two pose variables plus an azimuthal lighting angle. The goal is to discover, given only the unordered high-dimensional inputs, low-dimensional representations with coordinates that capture the intrinsic degrees of freedom of a data set. This problem is of prime importance not only in studies of vision, but also in speech, motor control, and a range of other physical and biological sciences. Because the data sets contain essential nonlinear structures, PCA and CMDS cannot truly reveal their structure. For example, both methods fail to detect the true degrees of freedom of the face data set in Fig. 8.6 and the intrinsic two dimensionality of the data set in Fig. 8.7. However, Isomap method displays these freedoms very well, for the geodesic distances reflect the true low-dimensional geometry of the manifold where the data resides. More details of the experiments are referred to [2].

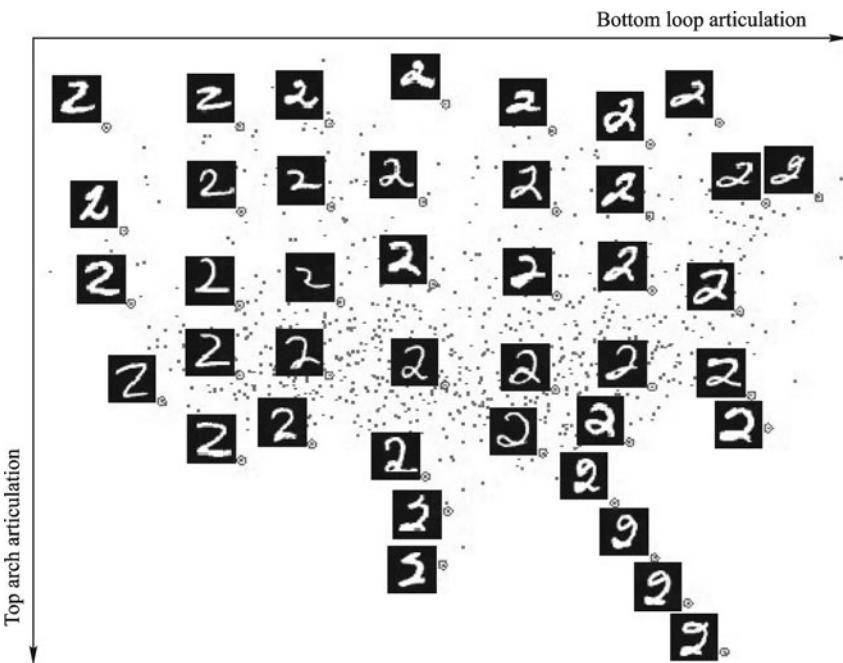


Fig. 8.7 Isomap applied to 1000 handwritten digit “2” from the MNIST database. Here Isomap embedding accurately shows the two most significant dimensions of the digit “2”: *bottom loop* (x -axis) and *top arch* (y -axis). Here ε -neighborhood (with $\varepsilon = 4.2$) is used because we do not expect a constant dimensionality to hold over the whole data set. Note: MNIST database is from (<http://yann.lecun.com/exdb/mnist/>). This graph is modified from the original in [2].

8.4.3 Conclusion

Isomap is a computationally stable algorithm. Various experiments show that most high-dimensional data lie on convex manifolds. Hence, Isomap method usually well models data and it is widely adopted in high-dimensional data analysis. The method is also insensitive to the neighborhood sizes used in the data graph construction, to the smoothness of the data underlying manifolds, and to the noise on the data. Hence, it is a promising DR method. However, estimating geodesic distances requires significantly intense computation. Since the Isomap kernel is dense, the Isomap algorithm is computationally expensive. Besides, if the data set does not lie on a convex manifold (for instance, the manifold has some “holes” or “cracks”), the Isomap method may not preserve the data geometry very well near the holes and cracks, producing large embedding errors there.

8.5 Justification of Isomap Methods

The key step in Isomap is the approximation of geodesic distance. In this section, we give the mathematical justification for the convergence of the approximation.

8.5.1 Graph Distance, S-distance, and Geodesic Distance

We first briefly review the definition of geodesic distance. For simplicity, we shall only discuss connected Riemannian manifold. A d -dimensional manifold M is called convex if there is a one-to-one differential map from M to \mathbb{R}^d such that the image of the manifold M is a convex region in \mathbb{R}^d .

Let \mathbf{p} and \mathbf{q} be two points on M . The geodesic distance between them is defined by

$$d_M(\mathbf{p}, \mathbf{q}) = \min_{\gamma \in \Gamma} \|\gamma\| = \int_a^b \|\dot{\gamma}(t)\| dt,$$

where Γ is the set of curves that connect \mathbf{p} and \mathbf{q} .

Since we only deal with finite sets that lie on hidden manifolds, we cannot directly compute the geodesic distances between the points in such a set. Utilizing the neighborhood structure on data, we compute the graph distances and use them to substitute the geometric distances in Isomap algorithm. Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ be a given finite data set and $G = [\mathcal{X}, E]$ be a connected graph on \mathcal{X} , where E is constructed by either ε -neighborhood or k -neighborhood. Recall that if the k -neighborhood is used, we need a modification to ensure that the graph is undirected. We first give a definition that recalls the notion of graph distance and also defines S -distance between two data points.

Definition 8.1. Let $G = [\mathcal{X}, E]$ be a given connected graph and \mathbf{x} and \mathbf{y} be two nodes of G . Then the G-distance (often called graph distance) and S -distance between \mathbf{x} and \mathbf{y} are defined by

$$\begin{aligned} d_G(\mathbf{x}, \mathbf{y}) &= \min_P (\|\mathbf{x}_1 - \mathbf{x}_0\| + \dots + \|\mathbf{x}_p - \mathbf{x}_{p-1}\|), \\ d_S(\mathbf{x}, \mathbf{y}) &= \min_P (d_M(\mathbf{x}_1, \mathbf{x}_0) + \dots + d_M(\mathbf{x}_p, \mathbf{x}_{p-1})), \end{aligned}$$

respectively, where $P = (\mathbf{x}_0, \dots, \mathbf{x}_p)$ denotes a path from $\mathbf{x} \stackrel{\text{def}}{=} \mathbf{x}_0$ to $\mathbf{y} \stackrel{\text{def}}{=} \mathbf{x}_p$.

We claim that G-distance and S -distance satisfy the distance axioms.

Lemma 8.1. Both G-distance and S -distance satisfy the distance axioms.

$$\begin{aligned} d_G(\mathbf{x}, \mathbf{y}) = 0 &\iff \mathbf{x} = \mathbf{y}, \quad d_G(\mathbf{x}, \mathbf{y}) = d_G(\mathbf{y}, \mathbf{x}), \\ d_G(\mathbf{x}, \mathbf{z}) &\leq d_G(\mathbf{x}, \mathbf{y}) + d_G(\mathbf{y}, \mathbf{z}), \end{aligned}$$

$$\begin{aligned} d_S(\mathbf{x}, \mathbf{y}) = 0 &\iff \mathbf{x} = \mathbf{y}, \quad d_S(\mathbf{x}, \mathbf{y}) = d_S(\mathbf{y}, \mathbf{x}), \\ d_S(\mathbf{x}, \mathbf{z}) &\leq d_S(\mathbf{x}, \mathbf{y}) + d_S(\mathbf{y}, \mathbf{z}). \end{aligned}$$

Furthermore,

$$d_M(\mathbf{x}, \mathbf{y}) \leq d_S(\mathbf{x}, \mathbf{y}), \quad d_G(\mathbf{x}, \mathbf{y}) \leq d_S(\mathbf{x}, \mathbf{y}).$$

Since the proof of the lemma is trivial, we skip it.

Recall that our goal is to prove that d_G approximates d_M when the Euclidean distance between two successive points in the path P is close to zero. Our strategy is to use d_S as a bridge between d_M and d_G . We shall show that d_M is not too smaller than d_S , and d_S is not too larger than d_G .

8.5.2 Relation between S-distance and Geodesic Distance

The following theorem is given in [11].

Theorem 8.2. *Let $G = [\mathcal{X}, E]$ be the graph defined in Definition 8.1 and M be the convex manifold, where \mathcal{X} resides. Let $\varepsilon > 0$ and $\delta > 0$ satisfy $4\delta < \varepsilon$. Assume that (i) if $d_M(\mathbf{x}, \mathbf{y}) \leq \varepsilon$, then the edge $(\mathbf{x}, \mathbf{y}) \in E$, (ii) for each $\mathbf{p} \in M$, there is $\mathbf{x} \in \mathcal{X}$ such that $d_M(\mathbf{x}, \mathbf{p}) \leq \delta$. Then for any $\mathbf{x}, \mathbf{y} \in \mathcal{X}$,*

$$d_M(\mathbf{x}, \mathbf{y}) \leq d_S(\mathbf{x}, \mathbf{y}) \leq (1 + 4\delta/\varepsilon)d_M(\mathbf{x}, \mathbf{y}). \quad (8.19)$$

Proof. The first inequality in (8.19) is obtained from Lemma 8.1. We only need to prove the second one in (8.19). Let γ be any piecewise-smooth arc that connects \mathbf{x} and \mathbf{y} . Write $\|\gamma\| = l$. If $l \leq \varepsilon - 2\delta$, then, by the assumption (i), $(\mathbf{x}, \mathbf{y}) \in E$ so that $d_S(\mathbf{x}, \mathbf{y}) = d_M(\mathbf{x}, \mathbf{y})$. If $l > \varepsilon - 2\delta$, we write $\mathbf{p}_0 = \mathbf{x}, \mathbf{p}_{s+1} = \mathbf{y}$, and cut up the arc γ into pieces by $\mathbf{p}_1, \dots, \mathbf{p}_s \in M$ ($\mathbf{p}_i \neq \mathbf{x}$ or \mathbf{y}) such that the arc lengths of $\gamma_i \stackrel{\text{def}}{=} \widehat{\mathbf{p}_i \mathbf{p}_{i+1}}, 0 \leq i \leq s$, satisfy

$$\begin{aligned} (\varepsilon - 2\delta)/2 &\leq \|\gamma_0\| \leq \varepsilon - 2\delta, \\ \|\gamma_i\| &\leq \varepsilon - 2\delta, \quad i = 1, \dots, s-1, \\ (\varepsilon - 2\delta)/2 &\leq \|\gamma_s\| \leq \varepsilon - 2\delta. \end{aligned}$$

By the assumption (ii), we can find $\mathbf{x}_i \in \mathcal{X}$ such that $d_M(\mathbf{x}_i, \mathbf{p}_i) \leq \delta, i = 1, \dots, s$. Since $\|\gamma_0\| \geq (\varepsilon - 2\delta)/2 > \delta$, $\mathbf{x}_1 \neq \mathbf{x}_0$ and $\mathbf{x}_s \neq \mathbf{x}_{s+1}$. It is easy to verify that $d_M(\mathbf{x}_i, \mathbf{x}_{i+1}) \leq \varepsilon$ so that $(\mathbf{x}_i, \mathbf{x}_{i+1}) \in E$. Therefore, we have

$$d_M(\mathbf{x}_0, \mathbf{x}_1) + \dots + d_M(\mathbf{x}_s, \mathbf{x}_{s+1}) \leq (s+1)\varepsilon \leq \frac{\varepsilon}{\varepsilon - 2\delta} \sum_{i=0}^s \|\gamma_i\| = \frac{\varepsilon}{\varepsilon - 2\delta} l.$$

Applying $\frac{\varepsilon}{\varepsilon - 2\delta} \leq 1 + 4\delta/\varepsilon$ and taking the infimum for γ , we finish the proof. \square

8.5.3 Relation between S-distance and Graph Distance

To prove that the S-distance d_S is not too much larger than the graph distance d_G is relatively complicated. It is only true if the manifold satisfies a few shape restrictions. To present the restrictions, we need the definition of the curvatures of a manifold. For an arc $\gamma(s)$ with the natural parameter s , we denote its *radius of curvature* at $\gamma(s)$ by $r(\gamma, s)$:

$$\frac{1}{r(\gamma, s)} = \|\ddot{\gamma}(s)\|.$$

Then the *minimum radius of curvature* of a manifold M is defined by

$$r_0 = r_0(M) = \min_{\gamma \in \Gamma(G)} (\min_s r(\gamma, s)),$$

where γ varies over all geodesic curves and s is taken over the domain of γ . The *minimum branch separation* of M is defined by

$$s_0 = s_0(M) = \arg \sup_{\mathbf{x}, \mathbf{y} \in M} \{s : \|\mathbf{x} - \mathbf{y}\| < s \Rightarrow d_M(x, y) < \pi r_0(M)\}.$$

Both $r_0(M)$ and $s_0(M)$ are positive numbers if M is a compact manifold. The following lemma shows the relation of the Euclidean distance and the geodesic distance between two points on M .

Lemma 8.2. *Let γ be a geodesic in M connecting two points \mathbf{x} and \mathbf{y} . If $d_M(\mathbf{x}, \mathbf{y}) < \pi r_0$, where r_0 is the minimum radius of curvature of M , then*

$$2r_0 \sin \left(\frac{d_M(\mathbf{x}, \mathbf{y})}{2r_0} \right) \leq d_2(\mathbf{x}, \mathbf{y}) \leq d_M(\mathbf{x}, \mathbf{y}). \quad (8.20)$$

Proof. The inequality on the right-hand side is a direct consequence of the fact that the shortest distance between two points is the line segment. The proof of the left-hand inequality is rather technical. We skip its proof. The readers are referred to Appendix of [12]. \square

Applying the inequality $\sin(t) \geq t - \frac{t^3}{6}$ to (8.20), we obtain

$$\left(1 - \frac{l^2}{24r_0^2}\right)l \leq d_2(\mathbf{x}, \mathbf{y}) \leq l, \quad (8.21)$$

where $l = d_M(\mathbf{x}, \mathbf{y})$. Hence, when l is small enough and M has positive $r_0(M)$, $d_2(\mathbf{x}, \mathbf{y})$ is close to $d_M(\mathbf{x}, \mathbf{y})$. From Lemma 8.2, we immediately obtain:

Corollary 8.1. *Let $\lambda > 0$ be given, $\mathbf{x} = \mathbf{x}_0$, and $\mathbf{x}_{n+1} = \mathbf{y}$. Assume that a path $\gamma = \{\mathbf{x}_0, \dots, \mathbf{x}_{n+1}\}$ from \mathbf{x} to \mathbf{y} in M satisfies the conditions*

$$\|\mathbf{x}_i - \mathbf{x}_{i-1}\| \leq s_0, \quad (8.22)$$

$$\|\mathbf{x}_i - \mathbf{x}_{i-1}\| \leq \frac{2}{\pi} \sqrt{24\lambda}, \quad (8.23)$$

for all $i, 1 \leq i \leq n + 1$. Then

$$(1 - \lambda)d_M(\mathbf{x}_i, \mathbf{x}_{i-1}) \leq d_2(\mathbf{x}_i, \mathbf{x}_{i-1}) \leq d_M(\mathbf{x}_i, \mathbf{x}_{i-1}), \quad (8.24)$$

which yields

$$(1 - \lambda)d_S(\mathbf{x}, \mathbf{y}) \leq d_G(\mathbf{x}, \mathbf{y}) \leq d_S(\mathbf{x}, \mathbf{y}). \quad (8.25)$$

Proof. Write $l = d_M(\mathbf{x}_i, \mathbf{x}_{i-1})$ and $d = d_2(\mathbf{x}_i, \mathbf{x}_{i-1})$. By the first assumption, we have $\frac{l}{2r_0} \leq \pi/2$. Applying $2t/\pi \leq \sin(t), \forall t \leq \pi/2$, and the second assumption above, we obtain

$$l \leq \frac{\pi}{2}d \leq r_0\sqrt{24\lambda},$$

which yields

$$(1 - \lambda) \leq \left(1 - \frac{l^2}{24r_0^2}\right).$$

By (8.21), we obtain (8.24). To prove (8.25), we first let γ be the path such that the path distance is equal to the graph distance:

$$\sum_{i=1}^{n+1} d_2(\mathbf{x}_{i-1}, \mathbf{x}_i) = d_G(\mathbf{x}, \mathbf{y}).$$

Then, we have

$$(1 - \lambda)d_S(\mathbf{x}, \mathbf{y}) \leq (1 - \lambda) \sum_{i=1}^{n+1} d_M(\mathbf{x}_{i-1}, \mathbf{x}_i) \leq \sum_{i=1}^{n+1} d_2(\mathbf{x}_{i-1}, \mathbf{x}_i) = d_G(\mathbf{x}, \mathbf{y}).$$

On the other hand, Let γ_S be the path from \mathbf{x} to \mathbf{y} , say $\{\mathbf{y}_0, \dots, \mathbf{y}_{n+1}\}$, with $\mathbf{y}_0 = \mathbf{x}$ and $\mathbf{y}_{n+1} = \mathbf{y}$, such that

$$\sum_{i=1}^{n+1} d_M(\mathbf{y}_{i-1}, \mathbf{y}_i) = d_S(\mathbf{x}, \mathbf{y}).$$

Then, we have

$$d_G(\mathbf{x}, \mathbf{y}) \leq \sum_{i=1}^{n+1} d_2(\mathbf{y}_{i-1}, \mathbf{y}_i) \leq \sum_{i=1}^{n+1} d_M(\mathbf{y}_{i-1}, \mathbf{y}_i) = d_S(\mathbf{x}, \mathbf{y}).$$

The proof is completed. \square

8.5.4 Main Result

We now can obtain the main result that confirms the approximation of graph distance to geodesic distance.

Theorem 8.3. Let $G = [\mathcal{X}, E]$ be a graph defined as in Definition 8.1 and $M \subset \mathbb{R}^D$ be a compact and convex manifold, where \mathcal{X} resides. For given $\varepsilon > 0, \delta > 0$, and two positive numbers $\lambda_1, \lambda_2 < 1$, with $4\delta < \lambda_2\varepsilon$, assume that the following conditions hold.

1. If $x, y \in \mathcal{X}$ satisfy $\|\mathbf{x} - \mathbf{y}\| \leq \varepsilon$, then the edge $(x, y) \in E$.
 2. For each $\mathbf{p} \in M$, there is $\mathbf{x} \in \mathcal{X}$ such that $d_M(\mathbf{x}, \mathbf{p}) \leq \delta$.
 3. $\max_{(\mathbf{x}, \mathbf{y}) \in E} \|\mathbf{x} - \mathbf{y}\| < \min(s_0(M), (2/\pi)r_0(M)\sqrt{24\lambda_1})$.
- Then, for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$, the following inequalities hold.

$$(1 - \lambda_1)d_M(\mathbf{x}, \mathbf{y}) \leq d_S(\mathbf{x}, \mathbf{y}) \leq (1 + \lambda_2)d_M(\mathbf{x}, \mathbf{y}). \quad (8.26)$$

Proof. Condition 1 implies that G contains all edges (\mathbf{x}, \mathbf{y}) with $d_M(\mathbf{x}, \mathbf{y}) \leq \varepsilon$. Applying Theorem 8.2 and using $4\delta < \lambda_2\varepsilon$, we obtain

$$d_G(\mathbf{x}, \mathbf{y}) \leq d_S(\mathbf{x}, \mathbf{y}) \leq (1 + \lambda_2)d_M(\mathbf{x}, \mathbf{y}),$$

which establishes the right-hand inequality of (8.26). To prove the left-hand one, we choose the path $\gamma = (\mathbf{x}_0, \dots, \mathbf{x}_{n+1})$ connecting $\mathbf{x} \stackrel{\text{def}}{=} \mathbf{x}_0$ and $\mathbf{y} \stackrel{\text{def}}{=} \mathbf{x}_{n+1}$ such that γ realizes the graph distance $d_G(\mathbf{x}, \mathbf{y})$. By Corollary 8.1, Conditions 2 and 3 imply that

$$(1 - \lambda_1)d_M(\mathbf{x}, \mathbf{y}) \leq (1 - \lambda_1)d_S(\mathbf{x}, \mathbf{y}) \leq d_G(\mathbf{x}, \mathbf{y}).$$

The theorem is proved. \square

References

- [1] de Silva, V., Tenenbaum, J.B.: Global versus local methods in nonlinear dimensionality reduction. In: S. Becker, S. Thrun, K. Obermayer (eds.) Neural Information Processing Systems (NIPS 2002), pp. 705–712. MIT Press (2002).
- [2] Tenenbaum, J.B., de Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science 290(5500), 2319–2323 (2000).
- [3] Bachmann, C.M., Ainsworth, T.L., Fusina, R.A.: Exploiting manifold geometry in hyperspectral imagery. IEEE Trans. Geo. Remote Sensing 43(3), 441–454 (2005).
- [4] Bachmann, C.M., Ainsworth, T.L., Fusina, R.A.: Improved manifold coordinate representations of large-scale hyperspectral scenes. IEEE Trans. Geo. Remote Sensing 44, 2786–2803 (2006).
- [5] Bachmann, C.M., Ainsworth, T.L., Fusina, R.A., Montes, M.J., Bowles, J.H., Korwan, D.R., Gillis, L.: Bathymetric retrieval from hyperspectral imagery using manifold coordinate representations. IEEE Trans. Geo. Remote Sensing 47, 884–897 (2009).
- [6] Balasubramanian, M., Schwartz, E.: The isomap algorithm and topological stability. Science 295, 9 (2002).
- [7] Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959).
- [8] Cailliez, F.: The analytical solution of the additive constant problem. Psychometrika 48(2), 305–308 (1983).

- [9] Kumar, V., Grama, A., Gupta, A., Karypis, G.: Introduction to Parallel Computing, Design and Analysis of Algorithms. The Benjamin/Cummings Publishing Company, Inc., California (1994).
- [10] Choi, H., Choi, S.: Robust kernel isomap. Pattern Recogn. 40.
- [11] Bernstein, M., de Silva, V., Langford, J., Tenenbaum, J.: Graph approximations to geodesics on embedded manifolds (2000).
- [12] Tenenbaum, J.B., de Silva, V., Langford, J.C.: Graph approximation to geodesics on embedded manifold (2000), Preprint.

Chapter 9 Maximum Variance Unfolding

Abstract Unlike Isomap method that preserves geodesic distances, MVU method learns the data from the similarities, preserving both local distances and angles between the pairs of all neighbors of each point in the data set. Since the method keeps the local maximum variance in dimensionality reduction processing, it is called maximum variance unfolding (MVU). Like multidimensional scaling (MDS), MVU can be applied to the cases that only the local similarities of objects in a set are given. In these cases, MVU tries to find a configuration that preserves the given similarities. Technically, MVU adopts semidefinite programming (SDP) to solve the DR problems. Hence, it is also called semidefinite embedding (SDE) or SDP. Solving a DR problem using MVU is expensive regarding both memory cost and time cost. To overcome the shortage of high computational cost, landmark MVU (LMVU) is introduced. In Section 9.1, we describe the MVU method and the corresponding maximization model. In Section 9.2, we give a brief review of SDP and introduce several popular SDP software packages. The experiments and applications of MVU are included in Section 9.3. The LMVU is discussed in Section 9.4.

9.1 MVU Method Description

The method of maximum variance unfolding (MVU), introduced by Weinberger and Saul [1, 2], is also called semidefinite embedding (SDE). It models dimensionality reduction problems using semidefinite programming (SDP). Isomap embedding preserves geodesic distances, which are close to Euclidean distances when the points are in a neighborhood. In Section 6.3, we already discussed the relation between Euclidean metric and centering Gram matrix. The Euclidean metric on a data set can be uniquely determined by the data Gram matrix up to shift and rotation. An entry of a Gram matrix is the inner product of a pair vectors. MVU method keeps the maximum variance in dimensionality reduction processing by preserving the local similarities represented by the local Gram matrix. The intuitive idea of the method is

illustrated in Fig. 9.1 and Fig. 9.2.

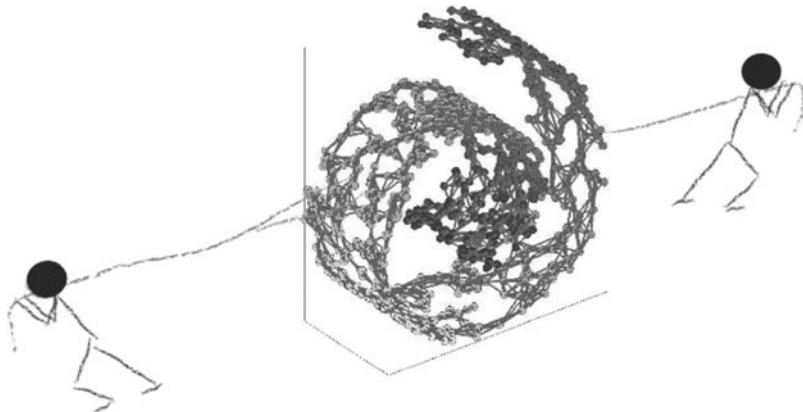


Fig. 9.1 A discrete manifold is revealed by forming the graph that pairwise connects each data point and its 6 nearest neighbors. This graph is modified from the original in [3].

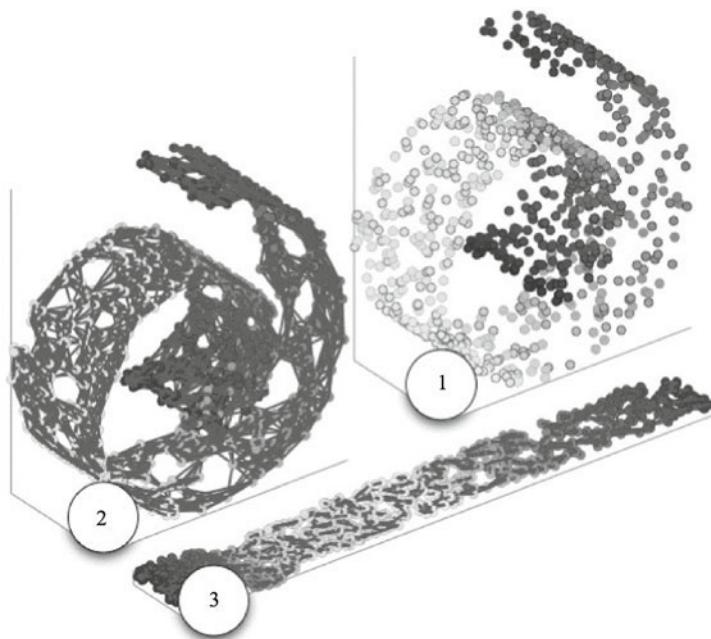


Fig. 9.2 In this figure, (1) 1000 points are sampled from a Swiss roll. (2) The Swiss roll is linearly reconstructed from 12 nearest neighbors for each point. (3) It is an embedding from MVU, with 4 nearest neighbors for each point. All of the distances between 5 points (4 neighbors and the point itself) are used in the constraints of MVU. This graph is modified from the original in [3], using a different setting in the DR processing.

9.1.1 Description of the MVU Method

As usual, we assume that the data set $\mathcal{X} \subset \mathbb{R}^D$ lies on a convex d -manifold M . Let h be a coordinate mapping on M such that for each $\mathbf{p} \in M$, $\mathbf{q} = h(\mathbf{p}) \in \mathbb{R}^d$ is the coordinate representation of \mathbf{p} . Then $g = h^{-1} : U \subset \mathbb{R}^d \rightarrow M$ is a parameterization of M . Without loss of generality, we may assume that $dh \stackrel{\text{def}}{=} dh_{\mathbf{p}}$ is isometric so that $dh \in \mathfrak{O}_{d,D}$ with $(dh)^T dh = I$ at each point $\mathbf{p} \in M$. We denote the image of \mathcal{X} under the mapping h by $\mathcal{Y} = h(\mathcal{X})$. Then \mathcal{Y} is the DR data set we want to find. Recall that \mathcal{Y} is assumed to have zero mean:

$$\sum_{j=1}^n y_{i,j} = 0, \quad i = 1, \dots, d. \quad (9.1)$$

We denote the Gram matrix of \mathcal{Y} by $\mathbf{K} = \mathbf{Y}'\mathbf{Y}$. Once \mathbf{K} is created, \mathbf{Y} can be obtained from the spectral decomposition of \mathbf{K} . To derive the kernel \mathbf{K} , MVU uses the following model.

Let P_i be a neighborhood of the point $\mathbf{x}_i \in \mathcal{X}$ and Q_i be its image $Q_i = h(P_i) \subset \mathbb{R}^d$. Since the neighborhood P_i is very close to the tangent hyperplane of M at \mathbf{x}_i , the local geometry of P_i resembles that of Q_i . More precisely, we have

$$\|\mathbf{x}_j - \mathbf{x}_k\| \approx \|\mathbf{y}_j - \mathbf{y}_k\|, \quad j, k \in N(i), \quad (9.2)$$

where $N(i)$ is the index set of the points in P_i . The MVU method formulates \mathcal{Y} as the solution of the following minimization problem:

$$\mathcal{Y} = \arg \min_{\mathcal{Y} \subset \mathbb{R}^d} \sum_{i=1}^n \sum_{j \sim k} (\|\mathbf{x}_k - \mathbf{x}_j\|^2 - \|\mathbf{y}_k - \mathbf{y}_j\|^2)^2, \quad (9.3)$$

where $k \sim j$ means that \mathbf{x}_k and \mathbf{x}_j are in the same neighborhood, say, in P_i . We also count \mathbf{x}_i in the relation “ \sim ” so that $k \sim i$ and $j \sim i$. The minimization problem (9.3) is not convex. Hence, it is hard to be solved directly. MVU then converts it to a semidefinite programming as follows.

Assume that \mathbf{K} is the Gram matrix of \mathbf{Y} . By the basic relation between centering Gram matrix and Euclidean metric presented in Section 6.3, we have

$$\|\mathbf{y}_j - \mathbf{y}_k\|^2 = \mathbf{K}_{k,k} + \mathbf{K}_{j,j} - 2\mathbf{K}_{k,j},$$

which yields the constraints

$$\mathbf{K}_{k,k} + \mathbf{K}_{j,j} - 2\mathbf{K}_{k,j} = \mathbf{S}_{k,j}, \quad j, k \in N(i), j > k, 1 \leq i \leq n, \quad (9.4)$$

where $\mathbf{S}_{k,j} = \|\mathbf{y}_j - \mathbf{y}_k\|^2$. Because \mathcal{Y} is centered, Equation (9.1) yields the following constraint on \mathbf{K} :

$$\sum_{i,j=1}^n \mathbf{K}_{i,j} = 0. \quad (9.5)$$

Finally, the data \mathcal{Y} is required to have the maximum variance:

$$\mathcal{Y} = \arg \max_{\mathcal{Y} \subset \mathbb{R}^d} \sum_{i,j=1}^n \|\mathbf{y}_i - \mathbf{y}_j\|^2 = \arg \max_{\mathcal{Y} \subset \mathbb{R}^d} \|\mathbf{D}_Y\|_F^2, \quad (9.6)$$

where \mathbf{D}_Y is the Euclidean metric on \mathcal{Y} . By maximizing (9.6), the points of \mathcal{Y} are pulled as far apart as possible, subject to the constraints (9.4) and (9.5). By Theorem 6.1 in Section 6.2, recalling that \mathbf{K} is the centering Gram matrix of \mathcal{Y} , we have

$$\frac{1}{2n} \|\mathbf{D}_Y\|_F^2 = \text{tr}(\mathbf{G}_Y^c) = \text{tr}(\mathbf{K}).$$

Therefore, the maximization problem (9.6) is finally converted to

$$\mathbf{K} = \arg \max_{K \in S\mathfrak{P}_n} \text{tr}(\mathbf{K}), \quad (9.7)$$

with the constraints (9.4) and (9.5). Combining them together, we obtain the maximization problem for the kernel K :

$$\text{maximize} \quad \text{tr}(\mathbf{K}), \quad \mathbf{K} \in \mathfrak{S}_n, \quad (9.8)$$

$$\text{s.t.} \quad \sum_{i,j=1}^n \mathbf{K}_{i,j} = 0, \quad (9.9)$$

$$\mathbf{K}_{k,k} + \mathbf{K}_{j,j} - 2\mathbf{K}_{k,j} = \mathbf{S}_{k,j}, \quad j, k \in N(i), \quad 1 \leq i \leq n, \quad (9.10)$$

$$\mathbf{K} \succeq 0, \quad (9.11)$$

where $\mathbf{K} \succeq 0$ is a short notation for \mathbf{K} being positive semidefinite.

Since the maximization problem for \mathbf{K} in (9.8) only involves the square-distance matrix \mathbf{S} , MVU algorithm also accepts the local distance matrix \mathbf{D} as the input. Hence, MVU can be applied to the applications where only the similarity relations of objects are available. In this sense, MVU is similar to MDS.

The data model in MVU method defines a special graphs on the data sets. The constraints (9.4) indicate that the local distances and angles between all points in the neighborhood of \mathbf{x} are preserved. Therefore, the points in the same neighborhood have to be mutually connected in the data graph. We shall call this type of graph a *locally complete graph*. In general, for a k -neighborhood or an ε -neighborhood, nodes are not mutually connected. A locally complete graph can be created by refining a usual graph $G = [\mathcal{X}, E]$. Let $G_r = [\mathcal{X}, E_r]$ denote the locally complete graph refined from G . Then $(k, j) \in E_r$ if and only if $(k, j) \in E$ or $(k, i), (j, i) \in E$ for some i . We illustrate the graph refinement in Fig. 9.3, where the left is a 2-degree regular graph, and the right is its refined locally complete graph.

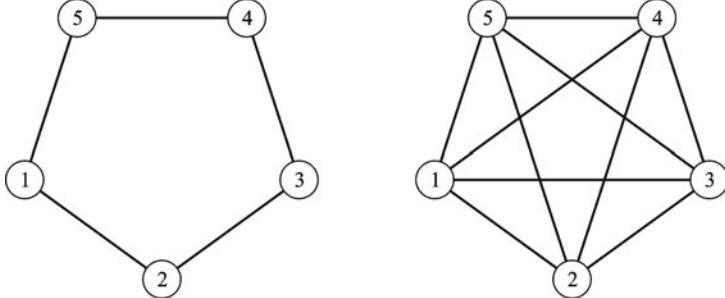


Fig. 9.3 Refined graph for MVU.

9.1.2 MVU Algorithm

Assume that an observed data set $\mathcal{X} \subset \mathbb{R}^D$ or a local distance matrix \mathbf{D} is given. MVU algorithm consists of 5 steps.

Step 1. Neighborhood definition. This step is necessary only if the input is the data set \mathcal{X} . This step is the same as the first step in the Isomap method. Either k -neighborhood or ε -neighborhood can be used to define the neighborhood system on the data set \mathcal{X} . In MVU, k -neighborhood is commonly used. We denote the corresponding graph by $G = [\mathcal{X}, \mathbf{A}]$. As described in the previous subsection, the adjacency matrix \mathbf{A} in MVU is only used to determine the indices of the constraints, i.e., a constraint of (9.4) will be made only if $\mathbf{A}(j, k) = 1$.

Step 2. Compute Gram matrix of observed data set. We may compute either the centering Gram matrix \mathbf{G}^c or the square-distance matrix \mathbf{S} to build the constraints of the semidefinite programming problem (9.8), depending the type of the input data. If the input is the data set \mathcal{X} , then the centering Gram matrix \mathbf{G} can be made. On the other hand, if the (local) square-distance matrix $\mathbf{S} = [s_{jk}]$ (or the local distance matrix $\mathbf{D} = [d_{jk}]$) is given, then the adjacency matrix $\mathbf{A} = [a_{jk}]$ is built by setting $a_{jk} = \text{sign}(s_{jk})$.

Step 3. Generate constraints for SDP. When $\mathbf{A}(k, j) = 1$, set the constraint

$$\mathbf{K}_{k,k} + \mathbf{K}_{j,j} - 2\mathbf{K}_{k,j} = \mathbf{G}_{k,k} + \mathbf{G}_{j,j} - 2\mathbf{G}_{k,j}$$

or

$$\mathbf{K}_{k,k} + \mathbf{K}_{j,j} - 2\mathbf{K}_{k,j} = s_{k,j}$$

for the SDP (9.8), depending on whether the Gram matrix \mathbf{G} or the square distance matrix \mathbf{S} is available. The centralization condition

$$\sum_{j=1}^n \mathbf{K}_{k,j} = 0$$

is also needed to add in the constraint set.

Step 4. Build MVU kernel using SDP. Apply an SDP algorithm, such as CSDP, SeDuMi, or SDTP3, to create the MVU kernel K by solving the semidefinite programming problem (9.8).

Step 5. Spectral decomposition of MVU kernel. Using a standard spectral decomposition algorithm to compute d -leading eigenvectors of K , which provide the dimensionality reduction data set Y . This step is the same as Step 4 in the Isomap algorithm.

The main step of the algorithm is Step 4, in which the SDP problem can be solved by an existent SDP package (see Subsection 9.2.1). We will not present Matlab code for MVU algorithm. The readers may refer to <http://www.cse.wustl.edu/~kilian/code/code.html>.

9.2 Semidefiniteness Programming

In this section, we briefly discuss the general formulation of a semidefinite programming(SDP). Let \mathbf{X} and \mathbf{C} be two $n \times n$ symmetric matrices, which are considered as the vectors in \mathbb{R}^{n^2} . We define their inner product by

$$\mathbf{C} \cdot \mathbf{X} = \sum_{i=1}^n \sum_{j=1}^n \mathbf{C}_{i,j} \mathbf{X}_{i,j}.$$

Thus, \mathbf{C} is a functional on the Euclidean space \mathbb{R}^{n^2} . Let $\mathbf{A}_i, 1 \leq i \leq m$, be m symmetric matrices in \mathfrak{S}_n and $\mathbf{b} = [b_1, \dots, b_m]' \in \mathbb{R}^m$. Then the following optimization problem is called a *semidefinite programming* (SDP):

$$\text{minimize } \mathbf{C} \cdot \mathbf{X} \tag{9.12}$$

$$\text{s.t. } \mathbf{A}_i \cdot \mathbf{X} = b_i, i = 1, \dots, m, \tag{9.13}$$

$$\mathbf{X} \succeq 0. \tag{9.14}$$

It is easy to see that the collection $S\mathfrak{P}_n$ is a convex set in \mathbb{R}^{n^2} and each constraint $\mathbf{A}_i \cdot \mathbf{X} = b_i$ is a hyperplane in \mathbb{R}^{n^2} . Assume that $\{\mathbf{A}_1, \dots, \mathbf{A}_m\}$ forms a linearly independent set in \mathbb{R}^{n^2} . Then the intersection of the m hyperplanes $\mathbf{A}_i \cdot \mathbf{X} = b_i, i = 1, \dots, m$, (denoted by S_m) is an $(n^2 - m)$ -dimensional linear manifold, which is called an *affine space*. Hence, the semidefiniteness programming problem (9.12) can be interpreted as to find the minimal value of the linear function $\mathbf{C} \cdot \mathbf{X}$ on the intersection of the convex set $X \succeq 0$ and the affine space S_m . The set of $n \times n$ symmetric matrices that satisfy the constraints is called a *feasible set* of SDP. An SDP has a solution if the feasible set is nonempty.

In MVU, the kernel \mathbf{K} is the solution of the SDP optimization problem (9.8) with the particular setting

$$\mathbf{C} = -\mathbf{I},$$

$$\begin{aligned}\mathbf{A}_i &\stackrel{\text{def}}{=} \mathbf{A}_{k,j}, \\ b_i &= S_{k,j},\end{aligned}$$

where the double index (k, j) is one-to-one mapped to the single index i ($1 \leq j, k \leq n$), and the binary matrix \mathbf{A}_i has only non-vanished entries at $(k, k), (j, j), (k, j), (j, k)$. Under this setting, the problem (9.8) always has a nonempty feasible set since the centering Gram matrix of \mathcal{X} is in the set.

A common method to solve SDP is the interior point method [4, 5]. The following packages are designed for solving SDP problems: SDPA, CSDP, SDPT3, SeDuMi, DSDP, PENSDP, SDPLR, and SBmeh. More detailed discussion on SDP software can be found in http://www-user.tu-chemnitz.de/~helmburg/sdp_software.html.

9.2.1 CSDP

CSDP is a software package developed by Borchers [6] for solving semidefinite programming problems. The current version is 6.1.0 for Windows. The algorithm is a predictor–corrector version of the primal–dual barrier method created by Helmburg, Rendl, Vanderbei, and Wolkowicz [7]. For a more detailed, but now somewhat outdated description of the algorithms in CSDP refer to [8]. CSDP is written in C for efficiency and portability. On systems with multiple processors and shared memory, CSDP can run in parallel. CSDP uses OpenMP directives in the C source code to tell the compiler how to parallelize various loops. The code is designed to make use of highly optimized linear algebra routines from the LAPACK and BLAS libraries.

CSDP also has a number of features that make it very flexible. CSDP can work with general symmetric matrices or with matrices that have defined block diagonal structure. CSDP is designed to handle constraint matrices with general sparse structure. The code takes advantage of this structure in efficiently constructing the system of equations that is solved at each iteration of the algorithm.

In addition to its default termination criteria, CSDP includes a feature that allows the user to terminate the solution process after any iteration. For example, this feature has been used within a branch and bound code for maximum independent set problems to terminate the bounding calculations as soon as a bound has been obtained that is good enough to fathom the current note. The library also contains routines for writing SDP problems and solutions to files and reading problems and solutions from files.

A stand alone solver program is included for solving SDP problems that have been written in the SDPA sparse format [9]. An interface to MATLAB and the open source MATLAB clone Octave is also provided. This interface can be used to solve problems that are in the format used by the SeDuMi [10].

The user's guide in [6] describes how to use the stand alone solver, MATLAB and Octave interface, and library routines. For detailed instructions on how to compile and install CSDP see the INSTALL file in the main directory of the CSDP software package. CSDP code in Matlab can be found in <https://projects.coin-or.org/Csdp/browser/matlab/csdp.m>.

9.2.2 SDPT3

SDPT3 (the current version is 4.0) is a software package developed by Toh, Tütüncü, and Todd designed to solve conic programming problems whose constraint cone is a product of semidefinite cones, second-order cones, non-negative orthants and Euclidean spaces, and whose objective function is the sum of linear functions and log barrier terms associated with the constraint cones. This includes the special case of determinant maximization problems with linear matrix inequalities. It employs an infeasible primal-dual predictor-corrector path-following method, with either the HKM or the NT search direction. The basic code is written in Matlab, but key subroutines in C are incorporated via Mex files. Routines are provided to read in problems in either SDPA or SeDuMi format. Sparsity and block diagonal structure are exploited. Low-rank structures are also exploited in the constraint matrices associated with the semidefinite blocks if such structures are explicitly given. To help the users use the software, the authors also include some examples to illustrate the coding of problem data for their SQLP solver. Various techniques to improve the efficiency and stability of the algorithm are incorporated. For example, step-lengths associated with semidefinite cones are calculated via the Lanczos method. Numerical experiments show that this general purpose code can solve more than 80% of a total of about 300 test problems with an accuracy of at least 10^{-6} in relative duality gap and infeasibility. The user's guide is in [11] and more details of the package are included in [12, 13] and the web site <http://www.math.nus.edu.sg/~mattohkc/sdpt3.html>.

9.3 Experiments and Applications of MVU

9.3.1 Testing MVU Algorithm on Artificial Surfaces

We first test MVU method with the same set of 4 artificial surfaces, 3D-cluster, Punched Sphere, Swissroll, S-curve, which are used in testing Isomap algorithm. In Fig. 9.4 we apply MVU to reduce their dimension, using 5 nearest points in the neighborhood construction. The experiment shows that

MVU works well for these surfaces, but the Swiss roll seems not truly unfolded perhaps because the DR data is not well scaled. The MVU method is not sensitive to the data smoothness. For example, although the data of 3D-cluster lack the smoothness on the line segments, MVU still works well for its DR.

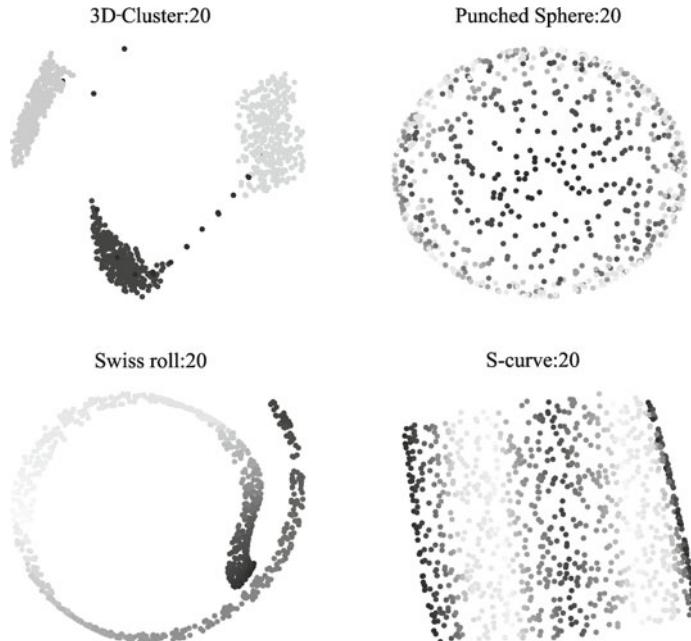


Fig. 9.4 MVU algorithm is tested on 4 surfaces: 3D-cluster, Punched sphere, Swiss roll, and S-curve. 1000 points are randomly selected from each of these surfaces. All of them are reduced to 2-dimensional data (on the plane) by the algorithm, using 5 neighbors of each point. The 2-dimensional DR data of these 4 surface data are shown in the figure. Top left: 3D-cluster. Top right: Punched sphere. Bottom left: Swiss Roll. Bottom right: S-Curve.

In Fig. 9.5, we display how the size of neighborhood impacts the results. When the size of neighborhood is reduced from 5 (that is used in Fig. 9.4) to 3, the algorithm detects the disconnection of the data graph for 3D-cluster. Then no output is produced for the data. The DR results for Punched sphere and S-curve are still satisfactory, while the result of Swiss Roll shows color mixture in some area. In general, when the size is too small, the data graph may not truly characterize the data similarity and therefore an incorrect dimensionality reduction data is produced. The relation between the size of neighborhood and the accuracy of the DR result is very complicated. A theoretical depiction is expected.

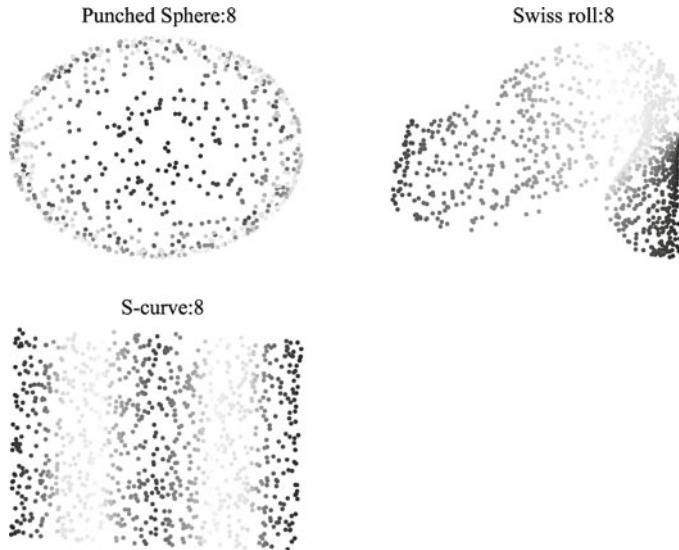


Fig. 9.5 The neighbors for each point are reduced to 3. The robust connection algorithm is not applied in the test. The results indicate that the neighborhood size is too small for 3D-cluster. Hence no valid DR result is produced. The DR results for Punched sphere and S-curve are still satisfactory, while DR data for Swiss roll show mixed colors in some area.

9.3.2 MVU Algorithm in Sensor Localization

The problem of sensor localization is formulated by Weinberger, Sha, Zhu, and Saul [14]. The problem can be illustrated by the example in Fig. 9.6. Imagine that sensors are located in major cities throughout the continental US, and that nearby sensors can estimate their distances to one another (e.g., via radio transmitters). From this unique local information, the problem of sensor localization is to compute the individual sensor locations and to identify the whole network topology. The problem can be viewed as computing a low rank embedding in two- or three-dimensional Euclidean space subject to local distance constraints. If the distances of all pairs are measured, then an inter-point Euclidean distance matrix is obtained and classical MDS can solve the problem. However, in the described situation, only the local information is available. The corresponding graph is not completed. Hence, it can be formulated into a nonlinear MDS problem: To find a configuration for a (sparse) local distance matrix.

Assume that there are n sensors distributed in the plane and the problem is formulated as an optimization over their planar coordinates $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\} \subset \mathbb{R}^2$. (Sensor localization in three-dimensional space can be solved in a similar way.) From (noisy) estimates of local pairwise distances

$[d_{ij}]_{i,j}$, the problem of sensor localization is to infer the planar coordinates \mathcal{Y} . Let a neighbor relation “ \sim ” be defined between sensors such that $i \sim j$ if the i th and j th sensors are sufficiently close, it is convenient to estimate their distance via limited-range radio transmission. The problems now can be formulated to minimize the sum-of-squares loss function [15] that penalizes large deviations from the estimated distances:

$$\min_{\mathcal{Y} \subset \mathbb{R}^2} \sum_{i \sim j} (\|\mathbf{y}_i - \mathbf{y}_j\|^2 - d_{ij}^2)^2. \quad (9.15)$$

This loss function is very similar to that we have discussed in CMDS, except that the sum in (9.15) is taken over the neighborhood system on the graph.

Note that in this type of problems, there is no original data set \mathcal{X} , but the local distance matrix.

In this example the authors of [14] consider the scenario where no “anchor points” are available by prior knowledge. In machine learning, it is called unsupervised learning problem. If the locations of a few sensors (“anchor points”) are known in advance, then it is called semi-supervised or supervised learning problem, which will not be discussed in this section. Our goal is to locate the sensors up to a global rotation, reflection, and translation, shortly, to find the configuration of the graph. Thus, to the above optimization, without loss of generality, the centralization constraint can be added:

$$\sum_{i=1}^n \mathbf{y}_i = 0, \quad (9.16)$$

which sets the geometric center of the sensor locations at the origin of the plane coordinate. Obviously, this constraint is only for mathematical convenience, getting nothing to do with the visualization.

Remark 9.1. It is straightforward to extend the approach to incorporate anchor points, leading to even better solutions. In this case, the centralization constraint (9.16) is not needed.

The optimization in (9.15) is not convex. To obtain a convex optimization of \mathcal{Y} , let $\mathbf{K} \stackrel{\text{def}}{=} [\langle \mathbf{y}_i, \mathbf{y}_j \rangle]$ and convert (9.15) together with the constraint (9.16) to the convex optimization problem

$$\text{Minimize: } \sum_{i \sim j} (\mathbf{K}_{ii} - 2\mathbf{K}_{ij} + \mathbf{K}_{jj} - d_{ij}^2)^2 \quad (9.17)$$

$$\text{subject to: } \sum_{i,j} \mathbf{K}_{ij} = 0, \quad \mathbf{K} \succeq 0, \quad (9.18)$$

where the first constraint in (9.18) is derived from (9.16), while the second constraint specifies that \mathbf{K} is positive semidefinite, which is necessary to interpret it as a Gram matrix in Euclidean space (see Chapter 6). In this

case, the vector set \mathcal{Y} is determined (up to rotation) by singular value decomposition of \mathbf{K} . The vector set \mathcal{Y} in the convex optimization (9.17) can generally lie in a subspace of the dimensionality equal to the rank of the kernel \mathbf{K} . To obtain planar coordinates, \mathcal{Y} can be further projected into a two dimensional subspace of maximum variance so that the planar coordinates are obtained from the top two eigenvectors of \mathbf{K} . Unfortunately, if the rank of \mathbf{K} is high, this projection loses information. Since the error of the projection grows with the rank of \mathbf{K} , a low rank is preferred. However, the rank of a matrix is not a convex function of its elements; thus the rank cannot be directly constrained as a part of the convex optimization. Hence, 2-dimensional visualization sometimes does not give an accurate topology of the object set.

Mindful of this problem, the approach to sensor localization used in [15] borrows an idea from the work on unsupervised learning [16, 17]. Assume that the sensors are centered on the origin. Then $\text{tr}(\mathbf{K}) = \sum_{i=1}^n \|\mathbf{y}_i\|^2$ represents the variance of \mathcal{Y} . Intuitively, a piece of paper has greater diameter than a crumpled one. Hence the term $\text{tr}(\mathbf{K})$ favors low-dimensional set \mathcal{Y} subject to the same constraints. Following this intuition, let the variance term be added to the loss function; thus, the minimization problem (9.17) is modified to

$$\text{Minimize: } \text{tr}(\mathbf{K}) - \nu \sum_{i,j} (\mathbf{K}_{ii} - 2\mathbf{K}_{ij} + \mathbf{K}_{jj} - d_{ij}^2)^2 \quad (9.19)$$

$$\text{subject to: } \sum_{i,j} \mathbf{K}_{ij} = 0, \quad \mathbf{K} \succeq 0, \quad (9.20)$$

where the parameter $\nu > 0$ balances the trade-off between maximizing variance and preserving local distances. Applying the Lagrange's method of multipliers to (9.8), the same minimization is obtained. Hence, the sensor localization problem (9.19) can be solved by MVU.

In [14], the authors used the following figures to illustrate how the sensor localization problem is solved by MVU. They did not assume any prior knowledge of sensor locations (e.g., no anchor points). The white noise is added to each local distance measurement with a standard deviation of 10% of the true local distance. There are sensor locations inferred for $n = 1055$ largest cities in the continental U.S.. On average, only 18 neighbors can be reached by each sensor. The original locations are shown in Fig. 9.6. In Fig. 9.7, the local distances between the sensor locations are measured, and represented in a graph, in which the nodes are sensor locations, and the edges (lines) connect nearest neighbors. The created graph is weighted by the local distances. Figure 9.8 displays the 2-D configuration of the graph in Fig. 9.7. It can be seen that the relative sensor locations in Fig. 9.8 are very close to the relative locations in Fig. 9.6.

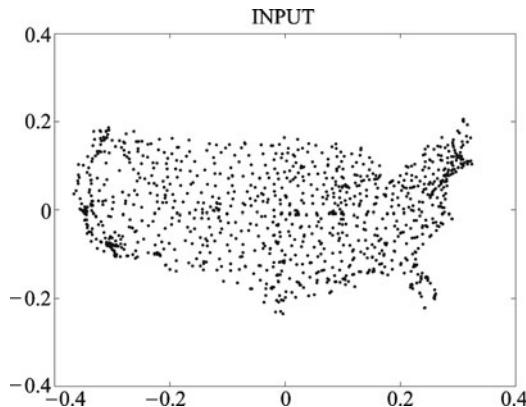


Fig. 9.6 Sensor locations inferred for $n = 1055$ largest cities in the continental US. On average, each sensor estimated local distances to 18 neighbors, with measurements corrupted by 10% Gaussian noise. We assume that the locations in the figure are not known in prior. Only the distance of two locations within radius of 0.1 can be measured. The figure is copied from [14].

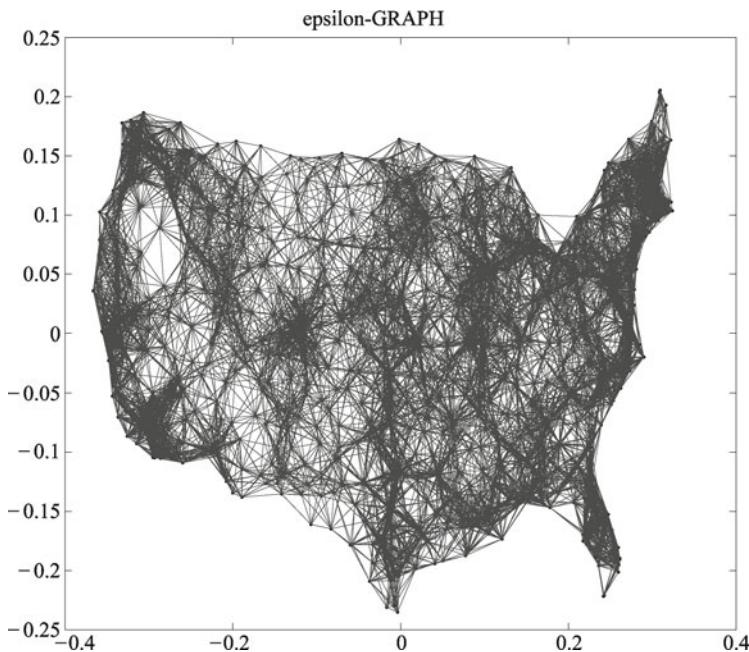


Fig. 9.7 The data graph shows the measured local distances between the 1055 sensors. The nodes in the graph are the sensors and the lengths of edge are roughly proportional to the distances serving as the weights of the data graph, which is used to generate the kernel of UVM. The figure is copied from [14].

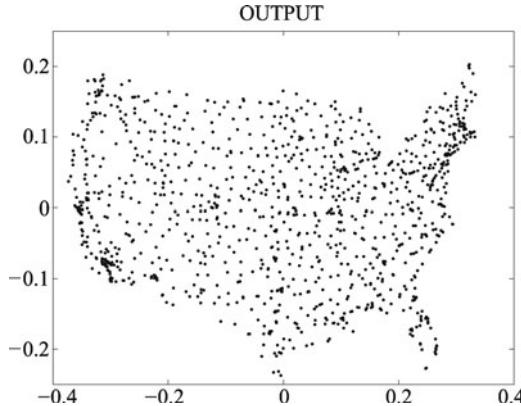


Fig. 9.8 The relative sensor locations are obtained by MVU using the 10 bottom eigenvectors of the graph Laplacian. Using PCA they are reduced to 2-D, and then the sensor locations are further corrected by conjugate gradient descent. The figure is copied from [14].

9.4 Landmark MVU

9.4.1 Description of Landmark MVU

One of the major problems with MVU is that it requires the use of semidefinite programming, which is computationally expensive. For a data set with N sample vectors, using k nearest points, the semidefinite programming employs total $k(k - 1)/2 \times N$ constraints in (9.4). When N increases, the number of constraints becomes too large to run the algorithm effectively. MVU is therefore limited to no more than 2000 points which use no more than 6 neighbors. If these bounds are breached, then the runtime of the algorithm becomes unreasonably long. This limitation has become more pressing recently, with the increasing availability of very large data sets and the corresponding increasing need for scalable dimensionality reduction algorithms. Landmark MVU (LMVU) was introduced in [18] to overcome the difficulty caused by large size samples. Landmark MVU technique first applies a standard MVU algorithm on a subset of the samples called *landmarks*, and then extends the DR results on the landmarks to other samples. The technique can also be used in other areas, such as online machine learning [16, 19–21].

Landmark MVU is effectively online with respect to the introduction of new data points. After the landmark points are selected and the initial calculation is carried out, all other points are embedded independently from each

other using a fixed linear transformation. A global calculation is necessary only if the embedding coordinates are required to be aligned with the principal axes of the data. Landmark MVU merges manifold interpolation into positively semidefinite programming. The strategy of LMVU can be sketched as follows.

Let $\mathcal{X} \subset \mathbb{R}^D$ be the observed data set and $\mathcal{Y} \subset \mathbb{R}^d$ be the DR data set of \mathcal{X} , where $d \ll D$. From the set \mathcal{Y} we can find a subset \mathcal{Y}_m having m points, $d < m$, such that \mathcal{Y}_m spans \mathbb{R}^d . We denote the data matrix corresponding to \mathcal{Y}_m by \mathbf{U} . Then there is an $n \times m$ matrix \mathbf{Q} such that

$$\mathbf{Y} = \mathbf{U}\mathbf{Q}' . \quad (9.21)$$

The $m \times m$ matrix

$$\mathbf{L} = \mathbf{U}'\mathbf{U}$$

is a psd one. Hence,

$$\mathbf{K} = \mathbf{Q}\mathbf{L}\mathbf{Q}' \quad (9.22)$$

is a DR kernel, whose spectral decomposition produces the DR set \mathcal{Y} . The landmark MVU is the method that finds the DR set \mathcal{Y} via the constructions of the linear transformation \mathbf{Q} and the spd matrix \mathbf{L} .

Note that the input data of LMVU is the observed data set \mathcal{X} . Therefore, the matrices \mathbf{Q} and \mathbf{L} in (9.22) need to be computed from \mathcal{X} . Hence, an LMVU algorithm consists of the following steps. First, as usual, define a neighborhood system on the data \mathcal{X} , which is represented by a graph $G = [\mathcal{X}, \mathbf{A}]$, where \mathbf{A} is the adjacency matrix. Second, select a landmark set from \mathcal{X} . Third, find the linear transformation \mathbf{Q} which (approximately) recovers all points in \mathcal{X} by the linear combinations of landmarks. Therefore, once the DR data of landmarks are computed, the DR data of \mathcal{X} can be computed by (9.21) too. Fourth, construct the landmark MVU kernel \mathbf{L} and make the spectral decomposition of \mathbf{L} . The first step is standard. We already studied the construction of graph neighborhood in Chapter 3. In the following we shall only study the constructions of \mathbf{Q} and \mathbf{L} .

9.4.2 Linear Transformation from Landmarks to Data Set

We first study the construction of the linear transformation \mathbf{Q} . As usual, we assume that the data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ resides on a d -manifold $M \subset \mathbb{R}^D$, ($d < D$). Let h be the coordinate mapping on M so that the DR set $\mathcal{Y} = h(\mathcal{X})$ is a manifold coordinate representation of \mathcal{X} . Let $\mathcal{Y}_m = h(\mathcal{X}_m)$, where \mathcal{X}_m is the landmark set for LMVU. For simplicity, without loss of generality, we assume that the landmark set \mathcal{X}_m consists of the first m points of the data set \mathcal{X} . That is

$$\mathcal{X}_1 \stackrel{\text{def}}{=} \mathcal{X}_m = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}.$$

Otherwise, we can permute the data matrix \mathbf{X} to make it true. Let \mathbf{X}_1 denote the matrix corresponding to \mathcal{X}_1 and let \mathbf{Y}_1 denote the matrix corresponding to $\mathcal{Y}_1 \stackrel{\text{def}}{=} \mathcal{Y}_m$. Since we construct \mathbf{Q} based on \mathcal{X} , the relation of \mathcal{Y} and \mathcal{Y}_1 in (9.21) needs to be converted to a relation between \mathcal{X} and \mathcal{X}_1 . Write $\mathcal{X}_2 = \mathcal{X} \setminus \mathcal{X}_1$, $\mathcal{Y}_2 = \mathcal{Y} \setminus \mathcal{Y}_1$, and denote their corresponding matrices by \mathbf{X}_2 and \mathbf{Y}_2 respectively. By $\mathcal{Y} = h(\mathcal{X})$ and $\mathcal{Y}_1 = h(\mathcal{X}_1)$, we rewrite (9.21) as

$$h(\mathbf{X}) = h(\mathbf{X}_1)\mathbf{Q}'.$$

Setting

$$\begin{aligned}\mathbf{X} &= [\mathbf{X}_1 \ \mathbf{X}_2], \\ \mathbf{Y} = h(\mathbf{X}) &= [h(\mathbf{X}_1) \ h(\mathbf{X}_2)] = [\mathbf{Y}_1 \ \mathbf{Y}_2],\end{aligned}$$

and

$$\mathbf{Q}' = \begin{bmatrix} \hat{\mathbf{Q}}_1 & \hat{\mathbf{Q}}_2 \end{bmatrix},$$

we have

$$[\mathbf{Y}_1 \ \mathbf{Y}_2] = [\mathbf{Y}_1 \hat{\mathbf{Q}}_1 \ \mathbf{Y}_1 \hat{\mathbf{Q}}_2],$$

which yields $\hat{\mathbf{Q}}_1 = \mathbf{I}_m$ (where \mathbf{I}_m is the $m \times m$ identity) and

$$\mathbf{Y}_2 = \mathbf{Y}_1 \hat{\mathbf{Q}}_2. \quad (9.23)$$

Hence, to find \mathbf{Q} , we only need to construct $\hat{\mathbf{Q}}_2$. To do so, we consider the neighborhood structure on \mathcal{X} . Assume that the high dimensional data \mathcal{X} is well sampled on the manifold $M \subset \mathbb{R}^D$ and a consistent neighborhood system is defined on \mathcal{X} , which generates the data graph $G = [\mathcal{X}, \mathbf{A}]$. Since the neighborhood is well defined, we can approximately recover each $\mathbf{x}_i \in \mathcal{X}$ by a weighted sum of its neighbors. Denote by $O(i)$ the neighbors of \mathbf{x}_i , and denote by $N(i) = \{i_1, \dots, i_k\}$ the index set of the vectors in $O(i)$. Then each point $\mathbf{x}_i \in \mathcal{X}_2$ can be approximately represented as a weighted sum of its neighbors:

$$\mathbf{x}_i \approx \sum_{j \in O(i)} w_{ji} \mathbf{x}_j, \quad m+1 \leq i \leq n. \quad (9.24)$$

Since $\mathbf{x}_j \approx \mathbf{x}_i$ if $\mathbf{x}_j \in O(i)$, it is reasonable to require the weight sum to be unit:

$$\sum_{j \in O(i)} w_{ji} = 1. \quad (9.25)$$

We call the weights in (9.24) *reconstruction weights*. By (9.25), we rewrite (9.24) as

$$\sum_{j \in O(i)} w_{ji} \hat{\mathbf{x}}_j \approx 0, \quad m+1 \leq i \leq n,$$

where $\hat{\mathbf{x}}_j = \mathbf{x}_i - \mathbf{x}_j$. Therefore, the weights w_{ji} can be obtained as the solution of the minimization problem

$$[w_{ji}]_{j \in O(i)} = \arg \min_{\sum w_{ji} = 1} \left\| \sum_{j \in O(i)} w_{ji} \hat{\mathbf{x}}_j \right\|^2. \quad (9.26)$$

Applying Lagrange's method of multipliers to (9.26), we obtain

$$[w_{ji}]_{j \in O(i)} = \arg \min \left\| \sum_{j \in O(i)} \hat{\mathbf{x}}_j w_{ji} \right\|^2 - \lambda \sum_{j \in O(i)} w_{ji}. \quad (9.27)$$

Write $\mathbf{w}_i = [w_{i_1, i}, \dots, w_{i_k, i}]'$ and $A_i = \sum_{j \in O(i)} \hat{\mathbf{x}}_j \hat{\mathbf{x}}_j'$. Then Equation (9.26) leads to

$$A_i \mathbf{w}_i = \lambda \mathbf{1}, \quad \lambda = \frac{1}{\sum_{j \in O(i)} w_{ji}}. \quad (9.28)$$

The weights w_{ji} constitute an $n \times (n - m)$ matrix $\mathbf{W} = [w_{ji}]$.

We split the sum on the right-hand side of (9.24) into two sums,

$$\sum_{j \in O(i), j \leq m} w_{ji} \hat{\mathbf{x}}_j + \sum_{j \in O(i), j > m} w_{ji} \hat{\mathbf{x}}_j, \quad m + 1 \leq i \leq n, \quad (9.29)$$

in which the vectors in the first sum is in the set \mathcal{X}_1 and the vectors in the second sum is in the set \mathcal{X}_2 . Therefore, we can write (9.24) in the matrix form

$$\mathbf{X}_2 \approx \mathbf{X}_1 \mathbf{W}_1 + \mathbf{X}_2 \mathbf{W}_2, \quad (9.30)$$

where \mathbf{W}_1 is constituted by the weights in the first sum of (9.29) while \mathbf{W}_2 is constituted by the weights in the second sum. The approximation in (9.30) is the best under the norm in (9.26).

By (9.24), the approximation in (9.30) is local for each \mathbf{x}_j . Since the neighborhood system is well defined on the manifold M , we have

$$h(\mathbf{x}_i) \approx \sum_{j \in O(i)} w_{ji} h(\mathbf{x}_j), \quad m + 1 \leq i \leq n, \quad (9.31)$$

which yields

$$\mathbf{Y}_2 \approx \mathbf{Y}_1 \mathbf{W}_1 + \mathbf{Y}_2 \mathbf{W}_2, \quad (9.32)$$

or, equivalently,

$$\mathbf{Y}_2 (\mathbf{I} - \mathbf{W}_2) \approx \mathbf{Y}_1 \mathbf{W}_1. \quad (9.33)$$

Solving the equation directly, we get

$$\mathbf{Y}_2 \approx \mathbf{Y}_1 \mathbf{W}_1 (\mathbf{I} - \mathbf{W}_2)^-, \quad (9.34)$$

where $(\mathbf{I} - \mathbf{W}_2)^-$ is a general inverse of $\mathbf{I} - \mathbf{W}_2$. Comparing (9.34) with (9.23), we can compute $\hat{\mathbf{Q}}_2$ using the formula

$$\hat{\mathbf{Q}}_2 = \mathbf{W}_1 (\mathbf{I} - \mathbf{W}_2)^-, \quad (9.35)$$

where $(\mathbf{I} - \mathbf{W}_2)^-$ can be computed by the formula

$$(\mathbf{I} - \mathbf{W}_2)^- = (\mathbf{I} - \mathbf{W}_2)^T ((\mathbf{I} - \mathbf{W}_2)(\mathbf{I} - \mathbf{W}_2)^T)^{-1}.$$

9.4.3 Algorithm for Landmark Linear Transformation

According to the previous subsection, the algorithm for the construction of landmark linear transformation contains two steps. Firstly, create a neighborhood system on the data set \mathcal{X} . Secondly, for a given landmark set, compute the linear transformation matrix Q .

```

function Phi = ldmktrans(X, K, landmark)
% Construct linear transformation for landmarks.
% SYNTAX:
%   PHI = ldmktrans(X,K,landmark)
% INPUTS:
%   X: D x n data matrix, X(:,i) is i-th point.
%   K: number of nearest neighbors, default:12.
%   Landmark: Index array for landmarks. Default
%   is 40 randomly selected points.
% OUTPUT:
%   Phi : m x n matrix represents all points
%   by landmarks: Y=Y1*Phi, where Y1 is the
%   DR landmark data matrix.
% Example: Find landmark transformation
%           for Swiss roll data.
% N=1000;
% tt = (3*pi/2)*(1+2*rand(1,N));
% height = 21*rand(1,N);
% X = [tt.*cos(tt); height; tt.*sin(tt)];
% PHI= LDMKTRANS(X, 12);
%
% CALL: data_neighborhood
%
% Initialize inputs.
[D,n]= size(X);
if nargin<3
    % Set default number of landmarks at 40.
    landmark=find(randi(n,[1,n])<41);
    if nargin<2
        % Set default number of neighbors
        % for computing weights to 12.
        K=12;
    end
end
%
% Step 1: Construct data-graph.
fprintf(1,'--Construct data graph\n');
options.k = K;
options.symmetric = 0;
[D2, nghb] = data_neighborhood(X, options);
pp = 1:n;
% Extract landmarks from samples
pp(landmark)=[];
```

```

nldmk=length(landmark);
% Step 2: Construct reconstruction weight matrix
fprintf(1,'--Construct partial weight matrix\n');
fprintf(1,'If K>D, regularization is used\n');
W = sparse(n,n-nldmk);
% regularization if(K>D)
tol = 1e-3*(K>D);
for i=1:n-nldmk
    %shift nearest neighbors
    z=X(:,find(nghb(:,pp(i))))-repmat(X(:,pp(i)),1,K);
    C=z'*z;
    C=C+tol*trace(C)*eye(K,K);
    W(find(nghb(:,pp(i)),i)) = C\ones(K,1);
end
% Normorlize W
W = W./repmat(sum(W,2), 1, n-nldmk);
% Step 3: Construct linear transformatation. ;
fprintf(1,'--Construct linear transformatation.\n');
Phi=[eye(nldmk), W(1:nldmk,:); ...
      /(eye(n-nldmk)-W(nldmk+1:n,:))];
bb=[landmark,pp];
Phi(:,bb) = Phi;

```

9.4.4 Construction of Kernel of Landmark MVU

The kernel of landmark MVU is represented in (9.22). In the previous subsection, we already constructed the linear transformation \mathbf{Q} . Similar to the construction of a regular MVU kernel, we can compute the psd matrix \mathbf{L} in (9.22) by applying semidefinite programming to the landmark set. For convenience, we still assume that the landmark set is \mathcal{X}_1 , and write $\mathbf{K} = \mathbf{Q}\mathbf{L}\mathbf{Q}'$. Similar to the regular MVU model (9.8), the semidefinite programming for \mathbf{L} is the following maximization problem:

$$\text{maximize } \text{tr}(\mathbf{Q}\mathbf{L}\mathbf{Q}'), \quad \mathbf{L} \in S\mathfrak{P}_m, \quad (9.36)$$

$$\text{s.t. } \sum_{i,j=1}^n \mathbf{K}_{i,j} = 0, \quad (9.37)$$

$$\mathbf{K}_{k,k} + \mathbf{K}_{j,j} - 2\mathbf{K}_{k,j} = S_{k,j}, \quad j, k \in N(i), \quad 1 \leq i \leq m, \quad (9.38)$$

$$\mathbf{L} \succeq 0. \quad (9.39)$$

To find \mathbf{L} , the semidefinite programming (9.36) has only about $k(k-1)/2 \times m$ constraints. Since m is a small number, the computational cost for solving (9.36) is quite low.

9.4.5 Experiments of LMVU

In Fig. 9.9, we illustrate the validity of LMVU method for dimensionality reduction when the size of the data set is large. We test LMVU algorithm for the Swiss roll data with 2000 sample points. 3-neighborhood is used to construct the data graph. We randomly select 61 landmarks from 2000 samples to construct the kernel of LMVU. The number of constraints for the corresponding semidefinite programming is now reduced to 367 and the total running time is reduced to 0.49 min.

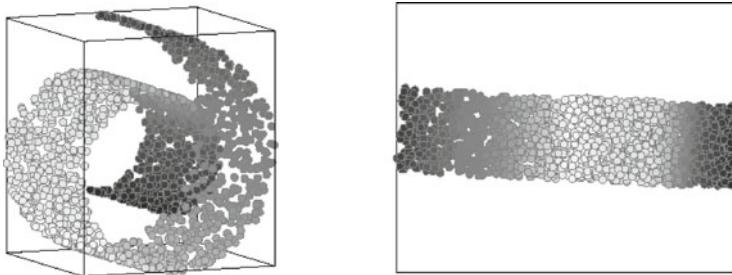


Fig. 9.9 Landmark MVU is applied to reducing the dimension of Swiss roll with 2000 sample points and 3 neighbors of each point. A regular MVU algorithm needs 30 minutes, by iterating 50 times for the DR. The total independent constraints in the SDP are 4316. The displayed result is obtained by LMVU with randomly selected 61 landmarks from 2000 samples and 3 neighbors of each point. It takes 52 times iterating, and number of independent constraints is 367. Total running time is only 0.49 min.

9.4.6 Conclusion

The MVU method has properties similar to those of Isomap method. It is computationally stable but the MVU DR kernel is dense and MVU algorithm is computationally intensive. MVU algorithm can be used to solve the problem, in which only the similarity matrix of data is available. In this sense, it can be considered as a nonlinear multidimensional scaling method. To reduce the computing cost, landmark MVU method is applied. The experiments show that LMVU performs fast and still yields satisfactory results in many cases.

References

- [1] Weinberger, K.Q., Packer, B.D., Saul, L.K.: Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. In: Proc. of the 10th International Workshop on AI and Statistics (2005).
- [2] Weinberger, K.Q., Saul, L.K.: Unsupervised learning of image manifolds by semidefinite programming. IEEE Conference on Computer Vision and Pattern Recognition 2, 988–995 (2004).
- [3] Saul, L.K., Roweis, S.T., Weinberger, K.Q., Sha, F., Packer, B.: Unfolding a manifold by semidefinite programming. Presentation, Computer and Information Science, University of Pennsylvania (2005).
- [4] Vandenberghe, L., Boyd, S.: Semidefinite programming. SIAM Review 38, 49–95 (1996).
- [5] Wright, S.: Primal-Dual Interior-Point Methods. PA: SIAM, Philadelphia (1997).
- [6] Borchers, B.: CSDP User's Guide (2009).
- [7] Helmberg, C., Rendl, F., Vanderbei, R.J., Wolkowicz, H.: An interior-point method for semidefinite programming. SIAM Journal on Optimization 6(2), 342–361 (1996).
- [8] Borchers, B.: CSDP, a C library for semidefinite programming. Optimization Methods & Software 11-2(1-4), 613–623 (1999).
- [9] Fujisawa, K., Kojima, M., Nakata, K., Yamashita, M.: SDPA (semidefinite programming algorithm) users manual - version 6.00. Tech. Rep. B-308, Tokyo Institute of Technology (1995).
- [10] Sturm, J.F.: Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. Optimization Methods & Software 11-2(1-4), 625–653 (1999).
- [11] Toh, K.C., Tütüncü, R.H., Todd, M.J.: On the implementation and usage of SDPT3—a Matlab software package for semidefinite-quadratic-linear programming (2006).
- [12] Toh, K., Todd, M., Tütüncü, R.: SDPT3—a Matlab software package for semidefinite programming. Optimization Methods and Software 11, 545–581 (1999).
- [13] Tütüncü, R., Toh, K., Todd, M.: Solving semidefinite-quadratic-linear programs using SDPT3. Mathematical Programming Ser. B 95, 189–217 (2003).
- [14] Weinberger, K.Q., Sha, F., Zhu, Q., Saul, L.K.: Graph Laplacian regularization for large-scale semidefinite programming. In: B. Schölkopf, J. Platt, T. Hofmann (eds.) Advances in Neural Information Processing Systems (NIPS), vol. 19. MIT Press, Cambridge, MA (2007).
- [15] Biswas, P., Liang, T.C., Toh, K.C., Wang, T.C., Ye, Y.: Semidefinite programming approaches for sensor network localization with noisy distance measurements. IEEE Transactions on Automation Science and Engineering 4(3), 360–371 (2006).
- [16] Liu, R., Jain, V., Zhang, H.: Sub-sampling for efficient spectral mesh processing. In: T. Nishita, Q. Peng, H.P. Seidel (eds.) Proc. 24th Comput. Graph. Intl. Conf., Lecture Notes in Comput. Sci., vol. 4035, pp. 172–184. Springer, Berlin (2006).
- [17] Weinberger, K.Q., Sha, F., Saul, L.K.: Learning a kernel matrix for nonlinear dimensionality reduction. In: the Twenty First International Conference on Machine Learning, Banff, Canada (2004).
- [18] de Silva, V., Tenenbaum, J.: Sparse multidimensional scaling using landmark points. Tech. rep., Stanford University (2004).

- [19] Belabbas, M.A., Wolfe, P.J.: On landmark selection and sampling in high-dimensional data analysis (2009). Submitted to Poyal Society, arXiv:0906.4582v1.
- [20] Bengio, Y., Paiement, J., Vincent, P., Delalleau, O., Roux, N.L., Ouimet, M.: Out-of-sample extensions for LLE, Isomap, MDS, Eigenmaps, and spectral clustering. In: S. Thrun, L. Saul, B. Schölkopf (eds.) Advances in Neural Information Processing Systems. MIT Press, Cambridge, MA (2004).
- [21] Platt, J.C.: Fastmap, MetricMap, and Landmark MDS are all Nyström algorithms. In: Proc. 10th Intl. Worksh. Artif. Intell. Statist., pp. 261–268 (2005).
- [22] Weinberger, K.Q., Saul, L.K.: An introduction to nonlinear dimensionality reduction by maximum variance unfolding. In: AAAI (2006).

Chapter 10 Locally Linear Embedding

Abstract In this chapter, locally linear embedding (LLE) method for dimensionality reduction is introduced. The method is based on simple geometric intuitions: If a data set is sampled from a smooth manifold, then neighbors of each point remain nearby and are similarly co-located in the low-dimensional space. In LLE, each point in the data set is linearly embedded into a locally linear patch of the manifold. Then low-dimensional data is constructed such that the locally linear relations of the original data are preserved. The chapter is organized as follows. In Section 10.1, we geometrically describe LLE method and its algorithm. The experiments of LLE are presented in Section 10.2 and some applications are introduced in Section 10.3. The mathematical justification of LLE is given in Section 10.4.

10.1 Description of Locally Linear Embedding

The method of locally linear embedding was introduced by Roweis and Saul [1, 2]. It is based on simple geometric intuitions. Assume that the data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ is well sampled from a smooth underlying manifold and a neighborhood structure is defined on the data set such that each neighborhood lies on or is close to a locally linear patch of the manifold. In LLE, the local geometry of each of these patches is characterized by linear coefficients that approximately reconstruct each data point from its neighbors. Then LLE algorithm computes a low-dimensional embedding that is optimized to preserve the local configurations of the data.

Although Isomap and MVU also adopt the local geometry of data, they construct the kernels globally, needing the data information beyond the neighborhood to keep the isometry of the embedding. Hence, an entry of the DR kernel is not directly computed from the points in a single neighborhood. For example, in Isomap, the DR kernel is derived from a geodesic metric whose entries are geodesic distances between pairs of points of the data set. Hence, to construct an Isomap DR kernel, we have to compute the geodesic distances between all pairs of points in the data set. Similarly, in MVU the

maximum variance is computed on the whole data set. Therefore, to construct an MVU DR kernel, we have to solve a global optimization problem. As a result, both of Isomaps and MVU produce dense DR kernels. On the contrary, LLE constructs the DR kernel directly from the local geometry of the data set, producing a sparse kernel.

10.1.1 Barycentric Coordinates

In LLE, a low-dimensional embedding is constructed to preserve the local geometry of data in a neighborhood, using the information within the local area. LLE realizes the local linear embedding using *barycentric coordinates* of each neighborhood. Barycentric coordinates are widely used in physics, computer graphics, and other areas. Assume that a set $\mathcal{P} = \{\mathbf{p}_1, \dots, \mathbf{p}_{m+1}\} \subset \mathbb{R}^m$ generates a simplex in \mathbb{R}^m with the vertices $\mathbf{p}_1, \dots, \mathbf{p}_{m+1}$. Then each point \mathbf{p} inside the simplex \mathcal{P} can be uniquely represented by the weights (or coefficients) w_1, \dots, w_{m+1} , as

$$\mathbf{p} = w_1\mathbf{p}_1 + w_2\mathbf{p}_2 + \dots + w_{m+1}\mathbf{p}_{m+1}$$

under the constraint $\sum_{i=1}^{m+1} w_i = 1$. The coefficients w_1, \dots, w_{m+1} are called the *barycentric coordinates* of \mathbf{p} (with respect to the simplex \mathcal{P}).

The advantage of barycentric coordinates is evident. The coordinates are “purely” geometric: They are independent of the physical locations and the dimension of the vertices. More precisely, barycentric coordinates are invariant under rigid motion and isometric embedding. If we consider each barycentric coordinate w_i as a function of \mathbf{p} : $w_i = w_i(\mathbf{p})$, then $w_i(\mathbf{p})$ has the following properties.

1. **Linearity.** It is a linear function of \mathbf{p} : $w_i(\lambda\mathbf{p} + (1 - \lambda)\mathbf{q}) = \lambda w_i(\mathbf{p}) + (1 - \lambda)w_i(\mathbf{q})$.
2. **Positivity.** If \mathbf{p} is strictly inside the simplex, then $w_i(\mathbf{p}) > 0$.
3. **Lagrange interpolating property.** For each vertex \mathbf{p}_j , $w_i(\mathbf{p}_j) = \delta_{i,j}$, where $\delta_{i,j}$ is the Kronecker delta.

Since the barycentric coordinate w_i favors \mathbf{p} that is close to \mathbf{p}_i , it naturally describes the local similarities of data. Due to these properties, under the barycentric coordinates each linear function F on \mathbf{p} has the representation

$$F(\mathbf{p}) = \sum_{i=1}^{m+1} w_i(\mathbf{p})f_i,$$

where $f_i = F(\mathbf{p}_i)$.

Therefore, if ϕ is a full-rank linear transformation, which maps the set $\{\mathbf{p}_i\}_{i=1}^{m+1}$ onto the set $\{\mathbf{q}_i\}_{i=1}^{m+1}$: $\mathbf{q}_i = \phi(\mathbf{p}_i)$, $1 \leq i \leq m+1$, then

$$\mathbf{q} \stackrel{\text{def}}{=} \phi(\mathbf{p}) = \sum_{i=1}^{m+1} w_i \mathbf{q}_i,$$

i.e., the barycentric coordinates are preserved under linear transformation.

We may generalize the barycentric coordinates to any convex polytope in \mathbb{R}^m . Assume that $\mathcal{P} = \{\mathbf{p}_1, \dots, \mathbf{p}_{s+1}\} \subset \mathbb{R}^m$ is a convex polytope in \mathbb{R}^m . Then each point \mathbf{p} inside the polytope can be represented as

$$\mathbf{p} = w_1\mathbf{p}_1 + w_2\mathbf{p}_2 + \dots + w_{s+1}\mathbf{p}_{s+1}$$

with the constraint $\sum_{i=1}^{s+1} w_i = 1$ and $w_i \geq 0, 1 \leq i \leq s+1$. In this case the coordinates may not be unique. Many methods, for example, Hermite mean value interpolation [3], can be used to compute the generalized barycentric coordinates.

10.1.2 LLE Method

We first focus on the construction of LLE DR kernels. Let $\mathbf{X}_{D \times n} = [\mathbf{x}_1 \dots \mathbf{x}_n]$ be the data matrix of a given data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$, which is sampled from a d -dimensional manifold $M \subset \mathbb{R}^D$. Assume that the neighborhood of each point of \mathcal{X} is well defined by using either k -nearest points or an ε -ball, even a hybrid one. Then the neighborhood structure defines a graph $G = [\mathcal{X}, \mathbf{A}]$ on \mathcal{X} , where \mathbf{A} is the adjacency matrix. Then the index set of the neighbors of \mathbf{x}_i , denoted by $N(i)$, is given by the i th row of \mathbf{A} such that

$$N(i) = \{j, \quad \mathbf{A}_{i,j} \neq 0\},$$

and the neighborhood of \mathbf{x}_i , denoted by O_i , is the set

$$O(i) = \{\mathbf{x}_j \in \mathcal{X}, \quad j \in N(i)\}.$$

Next, we construct an $n \times n$ weight matrix \mathbf{W} on the graph to define the similarities between the data points. Since a point $\mathbf{x}_j \notin O(i)$ is considered not similar to \mathbf{x}_i , we set the weight $w_{i,j} = 0$ if $\mathbf{A}_{i,j} = 0$. To define $w_{i,j}$ for each $j \in N(i)$, we choose the barycentric coordinate $w_{i,j}$ of \mathbf{x}_i with respect to its neighborhood $\{\mathbf{x}_j\}_{j \in O(i)}$, where \mathbf{x}_j is near \mathbf{x}_i on M . The geometric structure of $O(i)$ then can be approximated by its projection on the tangent space $T_{\mathbf{x}_i}$. Let f denote the o.g. projection from M to $T_{\mathbf{x}_i}$ and write $\tilde{\mathbf{x}} = f(\mathbf{x})$. The set $\{\tilde{\mathbf{x}}_j - \tilde{\mathbf{x}}_i, j \in N(i)\}$ spans a subspace of the tangent place of $T_{\mathbf{x}_i}$, and there is a weight set

$$\left\{ w_{i,j} \in \mathbb{R}; \quad j \in N(i), \quad \sum_{j \in N(i)} w_{i,j} = 1 \right\},$$

such that $\sum_{j \in N(i)} w_{i,j} \tilde{\mathbf{x}}_j = \tilde{\mathbf{x}}_i$. Note that some weights may be negative if $\tilde{\mathbf{x}}_i$ is not in the polytope of $\{\tilde{\mathbf{x}}_j\}_{j \in N(i)}$. We denote the graph degree at \mathbf{x}_i by $d_i = |N(i)|$ and denote the i th row of the weight matrix \mathbf{W} by $\vec{\mathbf{w}}_i =$

$[w_{i,1}, \dots, w_{i,n}]$, which is sparse, having at most d_i non-vanished entries. For convenience, we define the d_i dimensional sub-vector of $\vec{\mathbf{w}}_i$ by

$$\vec{\mathbf{w}}_i^{sub} = [\mathbf{w}_{i,1}, \dots, \mathbf{w}_{i,d_i}], \quad j_s \in N(i). \quad (10.1)$$

The set of sub-vectors $\{\vec{\mathbf{w}}_i^{sub}\}_{i=1}^n$ together with the neighbor-index set $\{N(i)\}_{i=1}^n$ uniquely determines the weight matrix \mathbf{W} . To avoid involving in the tangent projection f in the weight computation, the LLE method specifies the choice of the sub-vector $\vec{\mathbf{w}}_i^{sub}$ by minimizing the error

$$(\vec{\mathbf{w}}_i^{sub}) = \arg \min_{\overline{\mathbf{a}} \in \mathbb{R}^{d_i}} \left\| \mathbf{x}_i - \sum_{j \in N(i)} a_j \mathbf{x}_j \right\|, \quad \text{s.t.} \quad \sum_{j=1}^{d_i} a_j = 1, \quad (10.2)$$

where $\overline{\mathbf{a}} = [a_1, \dots, a_{d_i}]$. It is clear that the weight $w_{i,j}$ summarizes the contribution of \mathbf{x}_j to the reconstruction of \mathbf{x}_i . As Equation (10.2) shows, to compute the weights, we minimize the cost function in Equation (10.2) subject to two constraints:

Sparserness constraint. Each data point \mathbf{x}_i is reconstructed only from its neighbors $O(i)$, enforcing $w_{i,j} = 0$ if $\mathbf{x}_j \notin O(i)$.

Invariance constraint. The rows of the weight matrix sum to one:

$$\sum_{j=1}^n w_{i,j} = 1. \quad (10.3)$$

The invariance constraint is important to preserve the local geometry of the data. It indicates that 1 is an eigenvalue of \mathbf{W} and the eigenvector $\mathbf{1} = [1, 1, \dots, 1]'$ achieves 1. The mathematical explanation of its role will be given in Section 10.4. The sum

$$\sum_{j \in N(i)} w_{i,j} \mathbf{x}_j$$

is called the locally linear embedding (LLE) of \mathbf{x}_i .

Remark 10.1. Some entries in the weight matrix may be negative. If non-negative entries of \mathbf{W} are required, we may set each negative entry at zero and then re-normalize the row vectors of \mathbf{W} by the sum rule $\sum_{j=1}^n w_{i,j} = 1$. When \mathbf{W} is a non-negative matrix, 1 is the largest eigenvalue of \mathbf{W} . However, in applications, forcing the weights to become nonnegative does not always produce better results. Sometimes, it distorts the local geometric structure of data.

Finally, the DR data set $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\} \subset \mathbb{R}^d$ of LLE method is computed by optimally preserving the local linear embedding. That is, \mathcal{Y} minimizes the locally linear embedding error:

$$\mathcal{Y} = \arg \min_{\mathcal{Y} \in \mathbb{R}^d} \sum_{i=1}^n \left\| \mathbf{y}_i - \sum_{j \in N(i)} w_{i,j} \mathbf{y}_j \right\|^2 \quad (10.4)$$

under the constraints

$$\mathbf{Y}\mathbf{1} = \mathbf{0} \quad \text{and} \quad \mathbf{Y}\mathbf{Y}' = \mathbf{I}_d. \quad (10.5)$$

To solve the minimization problem (10.4) under the constraints (10.5), we construct the LLE DR kernel

$$\mathbf{K} = (\mathbf{I} - \mathbf{W})'(\mathbf{I} - \mathbf{W}), \quad (10.6)$$

which is positive semidefinite, having the 0-eigenvector $\mathbf{1}$.

Let $\mathbf{u}_1, \dots, \mathbf{u}_d$ denote the eigenvectors corresponding to the 2nd and up to the $(d+1)$ st smallest eigenvalues of \mathbf{K} , respectively. Write

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

Then $\mathbf{Y} = \mathbf{U}'_d$ is the DR data matrix. We put the justification in Section 10.4.

10.1.3 LLE Algorithm

An LLE algorithm consists of the following four steps.

Step 1. Neighborhood definition. The neighborhood of each point can be either k -neighborhood or ε -neighborhood. The number of points in a neighborhood is chosen to be greater than d . We denote by $G = [\mathcal{X}, \mathbf{A}]$ the constructed graph, where \mathbf{A} is the adjacency matrix. Recall that if ε -neighborhood is applied, then \mathbf{A} is symmetric, else if k -neighborhood is applied, then \mathbf{A} may be asymmetric. Because the value ε in ε -neighborhood definition depends on the dimension and the scale of the data points, ε -neighborhood is not often used in neighborhood construction for LLE.

Step 2. Weight matrix construction. A weight matrix $\mathbf{W} = [w_{i,j}]$ is generated on the graph $G = [\mathcal{X}, \mathbf{A}]$ in the following way. Set $w_{i,j} = 0$, if $\mathbf{A}_{i,j} = 0$, and compute $w_{i,j}$ by (10.2), if $\mathbf{A}_{i,j} = 1$. The equation (10.2) is solved by the least square method as follows. Let \mathbf{G}_i be the (shifted) Gram matrix of $O(i)$:

$$\mathbf{G}_i = [\langle \mathbf{x}_j - \mathbf{x}_i, \mathbf{x}_k - \mathbf{x}_i \rangle]_{j,k \in N(i)}.$$

We solve the equation

$$\mathbf{G}_i \vec{\mathbf{w}}_i^T = \begin{cases} \mathbf{1}, & \text{rank}(G_i) = d_i, \\ \mathbf{0}, & \text{rank}(G_i) < d_i, \end{cases}$$

and then normalize the solution by $\sum_{j \in N(i)} w_{i,j} = 1$.

Step 3. LLE kernel construction. Write $\mathbf{L} = \mathbf{I} - \mathbf{W}$, where \mathbf{I} is the $n \times n$ identity matrix. The LLE kernel is given by

$$\mathbf{K} = \mathbf{L}'\mathbf{L} = (\mathbf{I} - \mathbf{W})'(\mathbf{I} - \mathbf{W}).$$

Step 4. Eigen decomposition of LLE kernel. Let the spectral decomposition of \mathbf{K} be given by

$$\mathbf{K} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}',$$

where $\mathbf{\Lambda} = \text{diag}(\lambda_0, \lambda_1, \dots, \lambda_{n-1})$ with

$$0 = \lambda_0 \leq \lambda_1 \leq \dots \leq \lambda_{n-1}.$$

Assume that the graph $G = [\mathcal{X}, \mathbf{A}]$ is connected, then $\lambda_1 > 0$. Let $\mathbf{u}_1, \dots, \mathbf{u}_d$ be the 2nd to the $(d+1)$ st columns of \mathbf{U} . The matrix $\mathbf{Y} = [\mathbf{u}_1 \dots \mathbf{u}_d]'$ is the DR data matrix.

The Matlab code of LLE algorithm is presented in the following.

```

function [Y, ev] = drlle(X,d,options)
% Locally Linear Embedding for DR
% SYNTAX:
%   Y = drlle(X,d,options)
% INPUTS:
%   X: D x n data matrix, X(:,i) is ith point.
%   d: Dimension of output data.
% OPTIONS:
%   .epsilon: epsilon-neighbors.
%   .k: k-neighbors.
%   .nonnegativeW: True for nonnegative weights.
%   .verb: display the processing verbally.
% OUTPUT:
%   Y: d x n dimension-reduced data matrix.
%   ev: 1 - eigenvalues.
% Example: Reduce Swiss roll to 2-D.
%   N=1000;
%   tt = (3*pi/2)*(1+2*rand(1,N));
%   height = 21*rand(1,N);
%   X = [tt.*cos(tt); height; tt.*sin(tt)];
%   d=2;
%   options.k=10;
%   [Y,ev] = DRLLE(X, d, options);
%
% CALL: data_neighborhood
%
% Algorithm LLE refers to the paper
% S. T. Roweis and L. K. Saul,
% Nonlinear dimensionality reduction
% by locally linear embedding,
```

```
% Science 290, 2323-2326 (2000).
%
% Initialize inputs.
options.null=0;
[D,n] = size(X);
if ~isfield(options, 'epsilon')
    if isfield(options, 'k')
        K = options.k;
        if numel(K)==1
            K = K*ones(n,1);
        end
    else
        K = 10*ones(n,1);
    end
end
if isfield(options, 'nonnegativeW')
    nonnegativeW= options.nonnegativeW;
else
    nonnegativeW=false;
end
options.symmetric = 0;
% Step 1: Construct data-graph.
fprintf(1,'--Construct data graph\n');
% find # of neighbors for epsilon-neighborhood.
[D2, nghb] = data_neighborhood(X,options);
% Step2: Construct weight matrix
fprintf(1,'--Construct weight matrix\n');
fprintf(1,'If K>D, regularization is used\n');
W = sparse(n,n);
if isfield(options, 'epsilon')
    K = sum(nghb, 2);
end
tol = 1e-3.*(K > d);
for ii=1:n
    % shift ith point to origin
    z = X(:,nghb(:,ii))-repmat(X(:,ii),1,K(ii));
    C = z'*z;
    % regularization (K>D)
    C = C + tol(ii)*trace(C)*eye(K(ii),K(ii));
    W(ii,nghb(:,ii)) = C\ones(K(ii),1);
end
if nonnegativeW
    W = W*(W>0);
end
% Normorlize W
W = W./repmat(sum(W,2), 1, n);
% Step 3: Construct LLE kernel
fprintf(1,'--Construct LLE kernel.\n');
K = (speye(n)-W)'*(speye(n)-W);
```

```
% Step 4. Eigen decomposition
fprintf(1,'--Find dimension-reduced data.\n');
options.disp = 0; options.isreal = 1;
options.issym = 1;
[Y,eigv] = eigs(K,d+1,'SM',options);
% bottom eigenvector is [1,1,1,...] with eval 0
Y = Y(:,1:d)'/sqrt(n);
D = diag(eigv);
ev = 1- sqrt(D(1:d));
```

10.2 Experiments and Applications of LLE

10.2.1 Experiments on Artificial Surfaces

In Fig. 10.1, LLE algorithm is applied to reducing the 3-dimensional data of 4 artificial surfaces, 3D-cluster, Punched sphere, Swiss roll, and S-curve, to 2-dimensioanl data, where 10 nearest points are used to construct the neighborhood of a point. The experiment shows that LLE works well for these surfaces, except 3D-cluster. Note that the data of 3D-cluster lacks the surface

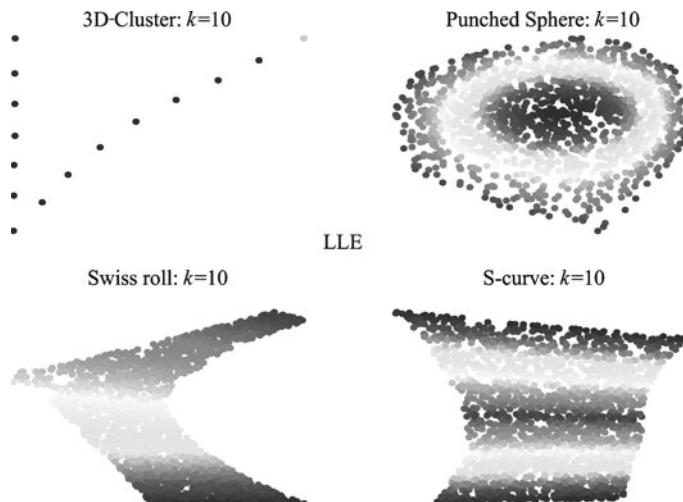


Fig. 10.1 LLE algorithm is tested on 4 surfaces: 3D-cluster, Punched sphere, Swiss roll, and S-curve. 1000 points are randomly sampled on each surface. All of them are reduced to 2-dimensional data (on the plane). The figure displays the DR results of these surfaces. The original 3-dimensional graphs are not displayed. They can be found in Section 1.5.

structure on the line segments. Because LLE is not an isometric mapping, it usually does not produce valid dimensionality reduction for the data such as 3D-cluster.

Noise on data impacts dimensionality reduction processing. In Fig. 10.2 and Fig. 10.3, we apply the LLE algorithm to noise data, with the noise standard deviation 0.2 and 0.4 respectively. When strong noise exists, the DR data may not preserve the data geometry.

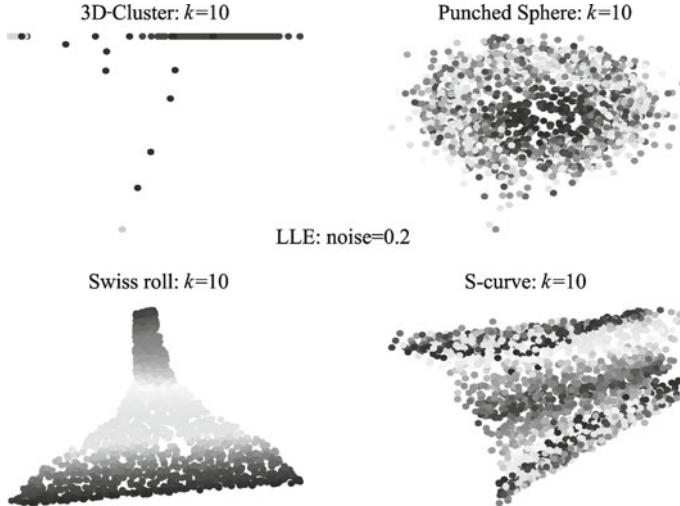


Fig. 10.2 LLE algorithm is applied on 4 surfaces with the noise standard deviation 0.2. All of these surfaces are sampled in the same way as in Fig. 10.1.

The results of LLE are stable over a range of neighborhood sizes. The stable range of the neighborhood sizes depends on the features of the data, such as the sampling density and the manifold geometry. Several criteria must be kept when choosing the number of neighbors of a data point. First, the algorithm can only be expected to recover embeddings whose dimensionality d is strictly less than the number of neighbors k . It is observed that some margin between d and k is generally necessary to obtain a topology-preserving embedding. However, finding the exact relation between k and d is very difficult. It depends on several facts, such as the data geometry and the distribution of samples. Note also that LLE algorithm is based on the assumption that a data point and its nearest neighbors can be modeled as locally linear. For curved data sets, choosing k too large will in general violate this assumption. As we mentioned above, when $k > d$, sometimes the weights are no longer uniquely defined. In this case, some further regularization must be added to break the degeneracy. Finally, the noise will also destroy the local linearity.

Figure 10.4 shows a range of embeddings discovered by the algorithm, all on the same data set, using different numbers of nearest neighbors. The

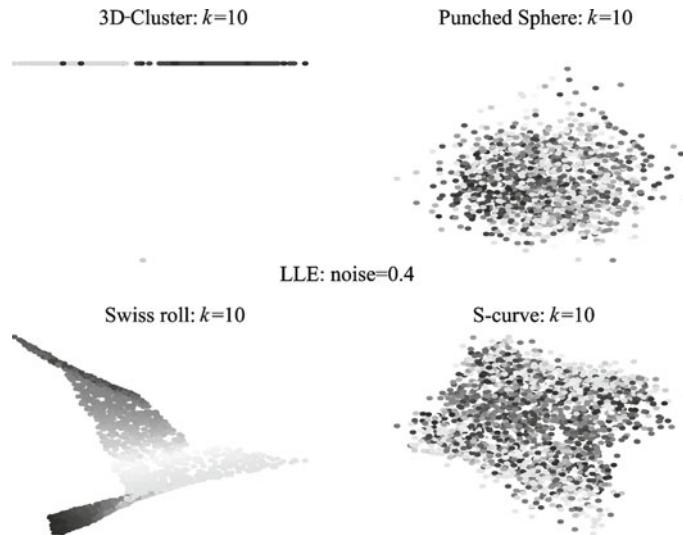


Fig. 10.3 LLE algorithm is applied on 4 surfaces with the noise standard deviation 0.4. All of these surfaces are sampled in the same way as in Fig. 10.1.

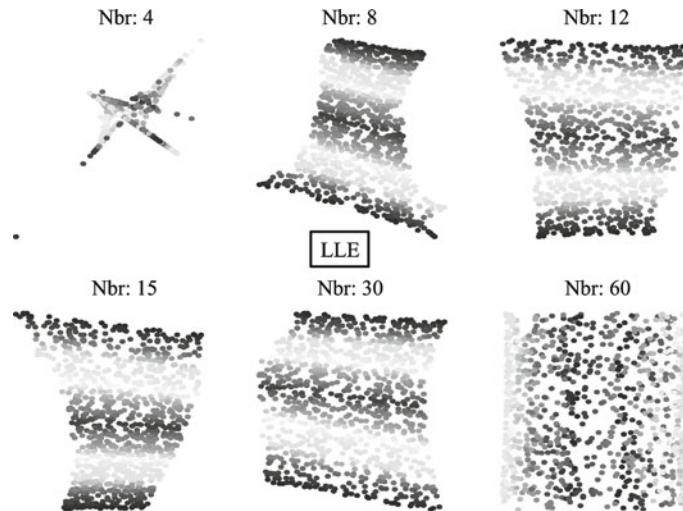


Fig. 10.4 Effect of neighborhood size on LLE. Embeddings of S-curve are computed for different numbers of nearest neighbors, 4, 8, 12, 15, 30, 60, respectively. Reliable embeddings from 3 dimension to 2 dimension are obtained over a wide range of values. If the number is too small (4 on top left) or too large (60 on bottom right), LLE does not succeed in preserving data geometry.

results are stable over a wide range of values but do break down as k becomes too small or too large.

The shapes of surfaces also impact the results. In Fig. 10.5, LLE algorithm is applied to Swiss rolls with lengths 21, 51, and 81, respectively. The number of nearest points, $k = 10$, is chosen for the LLE algorithm. The result is not very satisfied when the length of the roll is too long or too short.

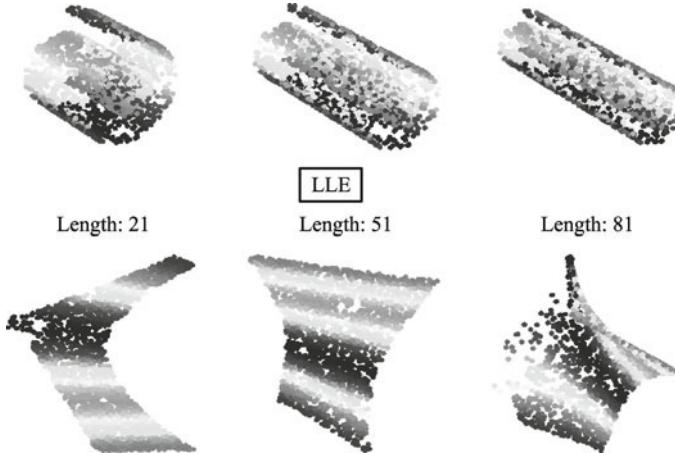


Fig. 10.5 Effect of surface shape on LLE. Embeddings of Swiss-Roll are computed for different choices of the lengths of the roll, 21, 51, and 81, respectively. It is shown that the dimensionality reduction for the Swiss Roll with length 51 is better than the other two.

10.2.2 Conclusion

Since the kernel of LLE is sparse, the LLE algorithm is fast. The method provides a fairly good linearization of the data on manifolds. However, the algorithm is occasionally computationally unstable. In the neighborhood definition, k -neighborhood is very common since it does not depend on the data scale and dimension. However, to determine an optimal k for a given data set is not easy. Small k usually leads to unstable computation and poor separation, while large k often causes misclassification. Fortunately, LLE algorithm adopts a wide range of the neighborhood sizes, so that the neighborhood size is not a very sensitive factor for LLE. If ε -neighborhood is applied, then it is hard to determine the suitable ε for a given data set since it is heavily dependent on the data scale and local geometry. Particularly, when the extrinsic data dimension is high, because of curse of high dimension, ε -neighborhood becomes unpractical method for defining data similarity.

10.3 Applications of LLE

10.3.1 LLE in Image Ordering

LLE method can be applied in image ordering when the parameters that determine images are nonlinearly related. Saul and Roweis [2] applied LLE to images of lips used in the animation of talking heads (Cosatto and Graf, 1998 [4]). This data set contained $N = 15960$ color (RGB) images of lips at 144×152 resolution ($D = 65664$). Figure 10.6 shows the first two components of these images discovered by LLE with $K = 24$ nearest neighbors. These components appear to capture highly nonlinear degrees of freedom associated with opening the mouth, pursing the lips, and clenching the teeth. Figure 10.7 shows how one particular neighborhood of lip images is embedded in this two-dimensional space. Dimensionality reduction of these images is useful for faster and more efficient animation. Particularly, the low-dimensional outputs of LLE can be used to index the original collection of high-dimensional images. Fast and accurate indexing is an essential component of example-based video synthesis from a large library of stored frames. The data set of lip images is the largest data set to which LLE has been applied by the authors of [2]. LLE scales relatively well to large data sets because it generates sparse intermediate results and eigen-problems.

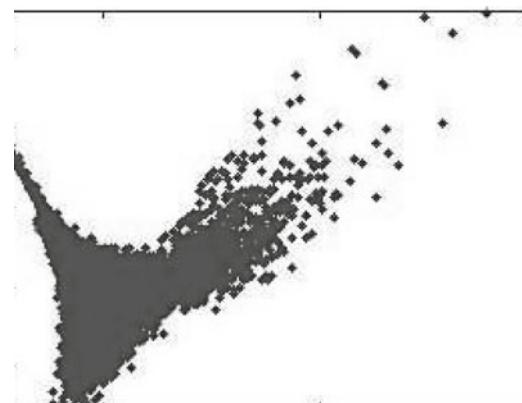


Fig. 10.6 High resolution ($D = 65664$) images of lips, mapped into the embedding space discovered by the first two coordinates of LLE, using $K = 24$ nearest neighbors. Representative lips are shown at different points in the space. The inset shows the first two LLE coordinates for the entire data set ($N = 15960$) without any corresponding images. The graph is copied from the original in [2].



Fig. 10.7 A typical neighborhood of $K = 24$ lip images mapped into the embedding space described by the first two coordinates of LLE. The figure shows a typical neighborhood in the overall space of lip images in Fig. 10.6. The graph is copied from the original in [2]

10.3.2 Supervised LLE

LLE is an unsupervised DR method, which does not need the structural knowledge of the data set in advance. However, as we mentioned above, sometimes it is hard to determine the optimal neighborhood size used in LLE. When the data have several clusters, i.e., they reside on a manifold having several unconnected submanifolds, the method usually does not work well. To extend the concept of LLE to multiple manifolds, the supervised LLE (SLLE) is introduced in [5] for this purpose.

Assume that a data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ is contained in multiple disjoint manifolds, corresponding to data classes. The classification information is partially known in advance: For each pair $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{X}$, it is known whether they are in the same class. We use the binary $n \times n$ matrix \mathbf{B} to represent the classification information: $\mathbf{B}_{ij} = 1$, if and only if \mathbf{x}_i and \mathbf{x}_j are in the same class. Assume that an adjacency matrix \mathbf{A}^e has been defined on \mathcal{X} in a regular way as in LLE. Let \mathbf{A}^o denote the intersection of \mathbf{A}^e and \mathbf{B} : $\mathbf{A}_{ij}^o = 1$ if and only if both $\mathbf{A}_{ij}^e = 1$ and $\mathbf{B}_{ij} = 1$. Let $t \in [0, 1]$ and \mathbf{W}^o and \mathbf{W}^e be the weight matrix corresponding to \mathbf{A}^o and \mathbf{A}^e respectively. Then the weight matrix for the t -SLLE is defined by

$$\mathbf{W} = (1 - t)\mathbf{W}^e + t\mathbf{W}^o, \quad 0 \leq t \leq 1. \quad (10.7)$$

It is clear that 0-SLLE is the regular LLE. We now briefly discuss 1-SLLE, in which $\mathbf{W} = \mathbf{W}^o (= [w_{ij}]_{i,j=1}^n)$ so that $w_{i,j} = 0$ if \mathbf{x}_i and \mathbf{x}_j are not in the same class. For simplicity, we assume that \mathbf{W} is symmetric and the points in the same class are connected in the graph $G = (\mathcal{X}, \mathbf{A}^0)$.

Theorem 10.1. Let $G^o = [\mathcal{X}, \mathbf{A}^o]$ be the graph on \mathcal{X} defined by an SLLE method with the neighborhood structure \mathbf{A}^o . Assume that for each pair \mathbf{x}_i and \mathbf{x}_j , if they are in the same class, there is a path in the graph G^o connecting them. Then \mathcal{X} has k different classes if and only if the zero eigenvalue of $\mathbf{L} = \mathbf{I} - \mathbf{W}^o$ has multiplicity k . Moreover, let $\mathbf{v}_1, \dots, \mathbf{v}_k$ be k binary o.g. 0-eigenvectors of \mathbf{L} . (That is, the components of these vectors are 0 and 1.) Then, each of them indicates one class in \mathcal{X} .

Proof. (Outline) The graph G^o has k -disconnected components. By Lemma 3.1 in Chapter 3, the zero eigenvalue of \mathbf{L} has multiplicity k . We can find a permutation \mathbf{P} such that

$$\mathbf{P}' \mathbf{L} \mathbf{P} = \begin{bmatrix} \mathbf{L}_1 & & & \\ & \mathbf{L}_2 & & \\ & & \ddots & \\ & & & \mathbf{L}_k \end{bmatrix}$$

is a block diagonal matrix in which the sum of each row is 0. Assume that the dimension of \mathbf{L}_j is $n_j \times n_j$. Define

$$\mathbf{1}_j = [\underbrace{0, \dots, 0}_{\sum_{s=1}^{j-1} n_s}, \underbrace{1, \dots, 1}_{n_j}, \underbrace{0, \dots, 0}_{\sum_{s=j+1}^k n_s}]' \in \mathbb{R}^n.$$

Then $\{\mathbf{1}_1, \dots, \mathbf{1}_k\}$ is an o.n. basis of the 0-eigenspace of $\mathbf{P}' \mathbf{L} \mathbf{P}$. It follows that $\{\mathbf{P}\mathbf{1}_1, \dots, \mathbf{P}\mathbf{1}_k\}$ is an o.n. basis of the 0-eigenspace of \mathbf{L} and all components of them are binary. They present the classification of k classes on \mathcal{X} . \square

10.4 Justification of LLE

10.4.1 Invariance Constraint

In Section 10.1, we made two constraints on the weight matrix \mathbf{W} . The sparseness constraint was discussed in Section 10.1 already. We now verify the invariance constraint (10.3). Intuitively, the weights ought to be invariant with respect to the chosen coordinates of \mathbb{R}^D . We now verify that the constraint (10.3) ensures the required invariance.

Theorem 10.2. The local weights of \mathbf{x}_i computed by (10.2) are invariant under rotations, translations, and dilations of the data set \mathcal{X} .

Proof. We denote the rotation operator by T and for $\lambda \neq 0$, denote by D_λ the dilation operator such that $D_\lambda \mathbf{x} = \lambda \mathbf{x}$. Then, by the linearity of T and

D_λ , we have

$$\sum_{j \in N(i)} w_{ij} (T\mathbf{x}_j) = T \left(\sum_{j \in N(i)} w_{ij} \mathbf{x}_j \right)$$

and

$$\sum_{j \in N(i)} w_{ij} (D_\lambda \mathbf{x}_j) = \lambda \left(\sum_{j \in N(i)} w_{ij} \mathbf{x}_j \right) = D_\lambda \left(\sum_{j \in N(i)} w_{ij} \mathbf{x}_j \right).$$

Let the weight set $\{w_{ij}^*\}_{j \in N(i)}$ be the solution of (10.2). Then, since T is an o.g. matrix,

$$\begin{aligned} (\tilde{w}_{ij}) &= \arg \min_{\sum w_{ij}=1} \left\| T\mathbf{x}_i - \sum_{j \in N(i)} w_{ij} T\mathbf{x}_j \right\| \\ &= \arg \min_{\sum w_{ij}=1} \left\| T \left(\mathbf{x}_i - \sum_{j \in N(i)} w_{ij} \mathbf{x}_j \right) \right\| \\ &= \arg \min_{\sum w_{ij}=1} \left\| \mathbf{x}_i - \sum_{j \in N(i)} w_{ij} \mathbf{x}_j \right\| = (w_{ij}^*). \end{aligned}$$

Similarly,

$$\begin{aligned} (\tilde{w}_{ij}) &= \arg \min_{\sum w_{ij}=1} \left\| \lambda \mathbf{x}_i - \sum_{j \in N(i)} w_{ij} \lambda \mathbf{x}_j \right\| \\ &= |\lambda| \arg \min_{\sum w_{ij}=1} \left\| \mathbf{x}_i - \sum_{j \in N(i)} w_{ij} \mathbf{x}_j \right\| = (w_{ij}^*). \end{aligned}$$

We now suppose that each vector in \mathbb{R}^D is translated by a vector $\mathbf{a} \in \mathbb{R}^D$. Then we have

$$\begin{aligned} \sum_{j \in N(i)} w_{ij} (\mathbf{x}_j - \mathbf{a}) &= \sum_{j \in N(i)} w_{ij} \mathbf{x}_j - \left(\sum_{j \in N(i)} w_{ij} \right) \mathbf{a} \\ &= \sum_{j \in N(i)} w_{ij} \mathbf{x}_j - \mathbf{a}, \end{aligned}$$

which yields

$$\begin{aligned}\tilde{w}_{ij} &= \arg \min_{\sum w_{ij}=1} \left\| (\mathbf{x}_i - \mathbf{a}) - \sum_{j \in N(i)} w_{ij} (\mathbf{x}_j - \mathbf{a}) \right\| \\ &= \arg \min_{\sum w_{ij}=1} \left\| \left(\mathbf{x}_i - \sum_{j \in N(i)} w_{ij} \mathbf{x}_j \right) \right\| = (w_{ij}^*) .\end{aligned}$$

The invariance of the LLE weights with respect to dilation, rotation, and shift is proved. \square

10.4.2 Condition for Weight Uniqueness

We now discuss the uniqueness of the solution for (10.2). Write

$$\mathcal{E}_i(\mathbf{W}) = \left\| \mathbf{x}_i - \sum_{j \in N(i)} w_{ij} \mathbf{x}_j \right\|^2 = \left\| \sum_{j \in N(i)} w_{ij} (\mathbf{x}_i - \mathbf{x}_j) \right\|^2$$

and set

$$\mathcal{E}(\mathbf{W}) \stackrel{\text{def}}{=} \sum_{i=1}^n \mathcal{E}_i(\mathbf{W}) = \sum_{i=1}^n \left\| \sum_{j \in N(i)} w_{ij} (\mathbf{x}_i - \mathbf{x}_j) \right\|^2 .$$

As usual, we denote by $\mathbf{1}$ the n -dimensional vector whose components are 1. Then the LLE weight matrix \mathbf{W}^* is the solution of the minimization problem:

$$\mathbf{W}^* = \arg \min_{\mathbf{W} \mathbf{1}=\mathbf{1}} \mathcal{E}(\mathbf{W}) ,$$

and this minimization problem can be uncoupled for each i , $1 \leq i \leq n$, as

$$\arg \min_{w_{ij}} \mathcal{E}_i(\mathbf{W}) , \quad \text{subject to } \sum_{j \in N(i)} w_{ij} = 1 . \quad (10.8)$$

Hence, the problem can be solved for each i individually. The variational approach to (10.8) leads to the Euler-Lagrange equation

$$\sum_{j \in N(i)} c_{sj} w_{ij} - \lambda = 0 , \quad s \in N(i) , \quad (10.9)$$

where λ is the Lagrange multiplier and $c_{sj} = \langle \mathbf{x}_j - \mathbf{x}_i, \mathbf{x}_s - \mathbf{x}_i \rangle$, $j, s \in N(i)$. Let k denote the cardinality of $N(i)$: $k \stackrel{\text{def}}{=} k(i) = |N(i)|$. By the neighborhood definition, $k > d$. Write $\mathbf{C} = (c_{sj})_{k \times k}$ and define $\vec{\mathbf{w}}_i^{sub} = [w_{ij_1}, \dots, w_{ij_k}]$ as in (10.1). Then (10.9) can be rewritten as the matrix equation

$$\mathbf{C}(\vec{\mathbf{w}}_i^{sub})^T = \lambda \mathbf{1} , \quad \mathbf{1} = [1, 1, \dots, 1]^T \in \mathbb{R}^k . \quad (10.10)$$

The uniqueness of solution of (10.10) depends on the rank of \mathbf{C} .

1. **C has full rank.** In this case, we solve for \vec{w}_i^{sub} by setting $\lambda = 1$, and then divide it by $\sum_{j \in N(i)} w_{ij}$ to normalize the weight vector. It is equivalent to the resetting of $\lambda = 1 / \sum_{j \in N(i)} w_{ij}$. In this case, the set $\{\mathbf{x}_j\}_{j \in N(i)}$ resides on a $(k - 1)$ -dimensional hyperplane $S \subset \mathbb{R}^D$ and \mathbf{x}_i does not reside on S . Let \tilde{x}_i be the projection of \mathbf{x}_i on S . Then the weights $\{w_{ij}\}_{j \in N(i)}$ are the barycentric coordinates of \tilde{x}_i with respect to the point set $\{\mathbf{x}_j\}_{j \in N(i)}$.
2. **The rank of C is equal to $k - 1$.** In this case, 0 is a simple eigenvalue of C . We then choose $(\vec{w}_i^{sub})^T$ to be the 0-eigenvector of C , normalized by $\sum_{j \in N(i)} w_{ij} = 1$. Under the constraint $\sum_{j \in N(i)} w_{ij} = 1$, the solution \vec{w}_i^{sub} is unique. In this case, the set $\{\mathbf{x}_j\}_{j \in N(i)}$ resides on a $(k - 1)$ -dimensional hyperplane S and \mathbf{x}_i resides on S too. Therefore, $\{w_{ij}\}_{j \in N(i)}$ are the barycentric coordinates of \mathbf{x}_i with respect to the point set $\{\mathbf{x}_j\}_{j \in N(i)}$.
3. **The rank of C is less than $k - 1$.** In this case, 0 is a multiple eigenvalue of C . We still can find a 0-eigenvector of C , which is normalized by $\sum_{j \in N(i)} w_{ij} = 1$. However, the solution now is not unique.

10.4.3 Reduction of the DR Data to LLE Model

Finally, we see how to find the DR data set \mathcal{Y} and what constraints it must satisfy. Note that the number of neighbors of each point of \mathcal{X} is greater than d . Hence, the graph $G = [\mathcal{X}, \mathbf{A}]$ has no isolate points. We also assume that G is connected so that 0 is a simple singular value of the Laplacian $\mathbf{L} = \mathbf{I} - \mathbf{W}$ and $\frac{1}{\sqrt{n}}\mathbf{1} \in \mathbb{R}^n$ is the 0-eigenvector. Let $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\} \subset \mathbb{R}^d$ be the DR data set of \mathcal{X} that is obtained as the solution of

$$\mathcal{Y} = \arg \min \Psi(\mathcal{Y}), \quad \Psi(\mathcal{Y}) = \sum_i \left\| \mathbf{y}_i - \sum_{j \in N(i)} w_{ij} \mathbf{y}_j \right\|^2. \quad (10.11)$$

Write $\mathbf{Y} = [\mathbf{y}_1 \dots \mathbf{y}_n] \in \mathfrak{M}_{d,n}$. Firstly, since the solution of the minimization problem (10.11) is invariant under the shift, we may ask the centroid of \mathcal{Y} to be at the origin: $\sum_{j=1}^n \mathbf{y}_j = 0$, which is equivalent to the constraint

$$\mathbf{Y}\mathbf{1} = 0. \quad (10.12)$$

Secondly, since $\mathbf{y}_i, 1 \leq i \leq n$, can be considered as the manifold coordinates of $\mathbf{x}_i \in \mathcal{X}$, we may ask all row vectors of \mathbf{Y} to form an o.n. system. It leads to the second constraint:

$$\mathbf{Y}\mathbf{Y}' = \mathbf{I}_d, \quad (10.13)$$

where \mathbf{I}_d is the $d \times d$ identity. We now solve the minimization problem under the constraints (10.12) and (10.13). Recall that

$$\begin{aligned}\Psi(\mathcal{Y}) &= \sum_{i=1}^n \left\langle \mathbf{y}_i - \sum_{j=1}^n w_{ij} \mathbf{y}_j, \mathbf{y}_i - \sum_{j=1}^n w_{ij} \mathbf{y}_j \right\rangle \\ &= \sum_{i=1}^n \left\langle \sum_{j=1}^n w_{ij} (\mathbf{y}_i - \mathbf{y}_j), \sum_{j=1}^n w_{ij} (\mathbf{y}_i - \mathbf{y}_j) \right\rangle,\end{aligned}$$

which yields

$$\Psi(\mathcal{Y}) = \text{tr}(\mathbf{Y} \mathbf{L}' \mathbf{L} \mathbf{Y}'), \quad (10.14)$$

where $\mathbf{L} = \mathbf{I} - \mathbf{W}$. Hence, by (10.14), the DR kernel is given by

$$\mathbf{K} = \mathbf{L}' \mathbf{L} = (\mathbf{I} - \mathbf{W})' (\mathbf{I} - \mathbf{W}).$$

Let the spectral decomposition of \mathbf{K} be given by

$$\mathbf{K} = \mathbf{U} \Lambda \mathbf{U}',$$

where $\Lambda = \text{diag}(\lambda_0, \lambda_1, \dots, \lambda_{n-1})$ with all eigenvalues arranged descended,

$$0 = \lambda_0 < \lambda_1 < \dots < \lambda_{n-1}.$$

We have

$$\Psi(\mathcal{Y}) = \text{tr}(\mathbf{Y} \mathbf{L}' \mathbf{L} \mathbf{Y}') = \text{tr}(\mathbf{Y} \mathbf{U} \Lambda (\mathbf{Y} \mathbf{U})').$$

Let $\mathbf{U}_d = [\mathbf{u}_1 \ \cdots \ \mathbf{u}_d]$ be the submatrix of \mathbf{U} , consisting of the eigenvectors corresponding to the 2nd to the $(d+1)$ st smallest eigenvalues of \mathbf{K} . Since $\mathbf{K} \mathbf{1} = 0$, the vector set $\{\mathbf{u}_1, \dots, \mathbf{u}_d\}$ satisfies the constraints (10.12) and (10.13). Then the minimum of $\Psi(\mathcal{Y})$ is attained by $\mathbf{Y} = \mathbf{U}_d'$.

References

- [1] Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. *Science* 290(5500), 2323–2326 (2000).
- [2] Saul, L.K., Roweis, S.T.: Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research* 4, 119–155 (2003).
- [3] Dyken, C., Floater, M.: Transfinite mean value interpolation. *Computer Aided Geometric Design* 26(1), 117–134 (2009).
- [4] Cosatto, E., Graf, H.P.: Sample-based synthesis of photo-realistic talking-heads. In: *Proceedings of Computer Animation*, IEEE Computer Society, pp. 103–110 (1998).
- [5] de Ridder, D., Kouropeteva, O., Okun, O., Pietikäinen, M., Duin, R.P.: Supervised locally linear embedding. In: *Proceedings of the 2003 joint international conference on Artificial neural networks and neural information processing*, pp. 333–341 (2003).

Chapter 11 Local Tangent Space Alignment

Abstract In this chapter, a dimensionality reduction embedding method, called local tangent space alignment (LTSA), is introduced. The method is based on the same geometric intuitions as LLE: If a data set is sampled from a smooth manifold, then the neighbors of each point remain nearby and similarly co-located in the low dimensional space. LTSA uses a different approach to the embedded space compared with LLE. In LLE, each point in the data set is linearly embedded into a locally linear patch of the manifold. Then low-dimensional data are constructed so that the locally linear relations of the original data are preserved. In LTSA, a locally linear patch is constructed by applying PCA on the neighbors. The patch then can be considered as an approximation of the tangent space at the point. Since the tangent place provides a coordinate representation of the neighbors, the coordinates give a low-dimensional representation of the patch. An alignment technique is introduced in LTSA to align the local representation to a global one. The chapter is organized as follows. In Section 11.1, we describe the method, paying more attention to the global alignment technique. In Section 11.2, the LTSA algorithm is developed. In Section 11.3, the experiments of LTSA are presented.

11.1 Description of Local Tangent Space Alignment

LTSA was introduced by Zhang and Zha [1]. Recent development and applications of this method can be found in [2–5].

11.1.1 Tangent Coordinates and Manifold Coordinates

To describe LTSA, we first review some basis knowledge in manifold calculus, which was introduced in Chapter 2. Let $M \subset \mathbb{R}^D$ be a d -dimensional manifold

and $f : U \subset \mathbb{R}^d \rightarrow M$ be a parameterization of M such that, for each $\mathbf{x} \in M$,

$$\mathbf{x} = f(\mathbf{y}), \quad \mathbf{y} \in U.$$

Let $h = f^{-1}$ be the coordinate mapping on M so that $\mathbf{y} = h(\mathbf{x})$, $\mathbf{x} \in M$, provides the manifold coordinates of \mathbf{x} . Assume that a given data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ lies on the manifold M and $\mathbf{y}_i = h(\mathbf{x}_i)$, $1 \leq i \leq n$. Then the coordinate set $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\} \subset \mathbb{R}^d$ is a dimensionality reduction of \mathcal{X} .

The LTSA method finds the DR data set \mathcal{Y} as follows. It is known that for each $\mathbf{p} \in M$, the local geometry of its neighborhood $O_{\mathbf{p}}$ can be learned from the local tangent place $T_{\mathbf{p}}M$. Let $Q_{\mathbf{p}}$ be the orthogonal project from $O_{\mathbf{p}}$ to the tangent place $T_{\mathbf{p}}M$. Then we have the approximation

$$Q_{\mathbf{p}}(\mathbf{x} - \mathbf{p}) \approx \mathbf{x} - \mathbf{p}, \quad \mathbf{x} \in O_{\mathbf{p}}. \quad (11.1)$$

Let $\mathcal{U} = \{\mathbf{u}_1, \dots, \mathbf{u}_d\}$ be an o.n. basis on $T_{\mathbf{p}}M$. Then, we have

$$Q_{\mathbf{p}}(\mathbf{x} - \mathbf{p}) = \sum_{i=1}^d \boldsymbol{\tau}_i \mathbf{u}_i, \quad \boldsymbol{\tau} = [\tau_1, \dots, \tau_d]' \in \mathbb{R}^d.$$

By the approximation (11.1), we also have

$$\mathbf{x} - \mathbf{p} \approx \sum_{i=1}^d \boldsymbol{\tau}_i \mathbf{u}_i, \quad (11.2)$$

which gives a (local) tangent coordinate representation of \mathbf{x} . On the other hand, each point $\mathbf{x} \in M$ has the (manifold) coordinate representation $\mathbf{x} = f(\mathbf{y})$, $\mathbf{y} \in \mathbb{R}^d$. Let $\mathbf{q} = h(\mathbf{p})$ and $df(\mathbf{q})$ be the derivative of f at \mathbf{q} . Applying Taylor's theorem, we have

$$\mathbf{x} - \mathbf{p} = df(\mathbf{q})(\mathbf{y} - \mathbf{q}) + O(d_2^2(\mathbf{y}, \mathbf{q})) \quad (11.3)$$

$$\approx df(\mathbf{q})(\mathbf{y} - \mathbf{q}), \quad (11.4)$$

where $df(\mathbf{q})$ is an invertible linear transformation. Equation (11.4) provides a (global) manifold coordinate representation of \mathbf{x} .

Equations (11.2) and (11.4) show that the manifold coordinate \mathbf{y} of the vector $\mathbf{x} \in \mathcal{X}$ can be obtained by aligning the local coordinate $\boldsymbol{\tau}$ of \mathbf{x} . The LTSA method realizes dimensionality reduction by finding the local coordinates for the given data, then aligning them with the manifold coordinates. The necessity of the alignment of local tangent coordinates is illustrated in Fig. 11.1.

Local Tangent space approximation

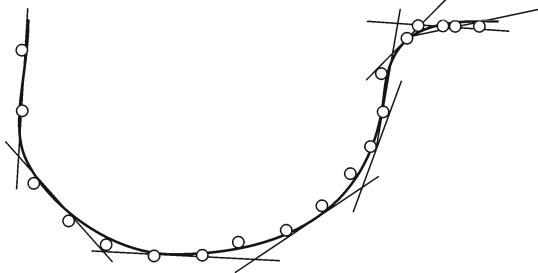


Fig. 11.1 At each point, tangent coordinates are used to represent the local geometry of the data. These coordinates need to align with a global representation. The graph is copied from the original in [1].

11.1.2 Local Coordinate Representation

In this subsection, we discuss the local coordinate representations of the points in the data set \mathcal{X} . Assume that on the data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset M \subset \mathbb{R}^D$ a neighborhood system is well defined and the system generates a graph $G = [\mathcal{X}, \mathbf{A}]$ on \mathcal{X} . We denote by $O(i)$ the neighborhood of \mathbf{x}_i and by $N(i) = \{i_1, \dots, i_k\}$ the index set of $O(i)$. The set $N(i)$ also corresponds to the non-zero entries in the i th row of \mathbf{A} . Let $k = |N(i)|$. We define $\bar{\mathbf{x}} \stackrel{\text{def}}{=} \frac{1}{k} \sum_{j \in N(i)} \mathbf{x}_j$, which is the geometric center of the point set $O(i)$. Although $\bar{\mathbf{x}}$ may not reside on M , it is very close to M . For simplicity, we assume that $\bar{\mathbf{x}} \in M$ (otherwise, use its nearest point in M instead). Denote by T_i the tangent space of M at $\bar{\mathbf{x}}$ and by H_i the tangent hyperplane $H_i = \bar{\mathbf{x}} + T_i$. Since the neighborhood system on \mathcal{X} is well defined, all points in O_i are very close to their projections on the hyperplane H_i . Let $F : O_i \rightarrow H_i$ be the o.g. projection such that

$$F(\mathbf{x}_j) = \bar{\mathbf{x}} + \mathbf{p}_j, \quad j \in N(i), \mathbf{p}_j \in T_i. \quad (11.5)$$

Due to the properties of linear approximation of T_i ,

$$\frac{1}{k} \sum_{j \in N(i)} \mathbf{p}_j = \frac{1}{k} \sum_{j \in N(i)} F\mathbf{x}_j - \bar{\mathbf{x}} \approx 0.$$

Hence, the vector set $\{\mathbf{p}_j\}_{j \in N(i)}$ is centered, so that

$$\mathbf{P}_i = \mathbf{P}_i \mathbf{H},$$

where \mathbf{H} is the $k \times k$ centralizing matrix and $\mathbf{P}_i = [\mathbf{p}_{i_1}, \dots, \mathbf{p}_{i_k}]$, where $i \in N(i)$. Assume that $k > d$ and the vector set $\{\mathbf{p}_j\}_{j \in N(i)}$ spans T_i . Then

we can construct an o.n. basis of T_i using the singular value decomposition of the matrix $\mathbf{P}_i = [\mathbf{p}_{i_1} \cdots \mathbf{p}_{i_k}]$:

$$\mathbf{P}_i = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}', \quad (11.6)$$

where the column vectors of $\mathbf{U} = [\mathbf{u}_1 \cdots, \mathbf{u}_d] \in \mathfrak{O}_{D,d}$ form an o.n. basis of T_i . By (11.6),

$$\boldsymbol{\Theta}_i \stackrel{\text{def}}{=} \boldsymbol{\Sigma} \mathbf{V}' = [\tau_{s,j}]_{s,j=1}^{d,k}$$

is the local coordinate matrix of \mathbf{P}_i and

$$\mathbf{p}_j = \sum_{s=1}^d \boldsymbol{\tau}_{s,j} \mathbf{u}_s, \quad j \in N(i). \quad (11.7)$$

In order to compute the local coordinates $\tau_{s,j}$, we replace $\{\mathbf{p}_j\}_{j \in N(i)}$ by its approximation $\mathcal{X}_i \stackrel{\text{def}}{=} \{\mathbf{x}_j - \bar{\mathbf{x}}\}_{j \in N(i)}$. Write $\mathbf{X}_i = [\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_k}]$ and let the singular value decomposition of the centered data matrix $\mathbf{X}_i \mathbf{H}$ be

$$\mathbf{X}_i \mathbf{H} = \mathbf{U}_D \boldsymbol{\Sigma}_D \mathbf{V}'_D, \quad (11.8)$$

where $\mathbf{U}_D = [\mathbf{u}_1 \cdots \mathbf{u}_D]$ is an o.n. matrix, $\boldsymbol{\Sigma}_D$ is a $D \times D$ diagonal matrix consisting of all singular values of $\mathbf{X}_i \mathbf{H}$, and $\mathbf{V}_D = [\mathbf{v}_1 \cdots \mathbf{v}_D] \in \mathfrak{O}_{k,D}$. Write $\mathbf{U}_d = [\mathbf{u}_1 \cdots \mathbf{u}_d]$, $\mathbf{V}_d = [\mathbf{v}_1 \cdots \mathbf{v}_d]$, and $\boldsymbol{\Sigma}_d = \text{diag}(\sigma_1, \dots, \sigma_d)$. By (11.5), $\mathbf{U}_d \boldsymbol{\Sigma}_d \mathbf{V}'_d$ is (approximatively) equal to \mathbf{P}_i in (11.6). For convenience, we shall delete the subscript d in $\mathbf{U}_d \boldsymbol{\Sigma}_d \mathbf{V}'_d$, identifying it with (11.6). Equation (11.8) leads to

$$\mathbf{X}_i \mathbf{H} = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}' (= \mathbf{U} \boldsymbol{\Theta}_i). \quad (11.9)$$

We now discuss the relation between the local coordinate representation of the set \mathcal{X}_i and its manifold representation. Applying (11.4) to the set \mathcal{X}_i with the setting $\mathbf{p} = \bar{\mathbf{x}}$, we have

$$\mathbf{x}_j - \bar{\mathbf{x}} = df(\bar{\mathbf{y}})(\mathbf{y}_j - \bar{\mathbf{y}}), \quad (11.10)$$

where $\bar{\mathbf{y}} = h(\bar{\mathbf{x}})$. If the approximation is accurate enough, we have $\bar{\mathbf{y}} = \sum_{j \in N(i)} \mathbf{y}_j$. Setting $\mathbf{Y}_i = [\mathbf{y}_{i_1} \cdots \mathbf{y}_{i_k}]$, we can rewrite (11.10) in the matrix form

$$\mathbf{X}_i \mathbf{H} = df(\bar{\mathbf{y}}) \mathbf{Y}_i \mathbf{H}. \quad (11.11)$$

Recall that $df(\bar{\mathbf{y}})$ is invertible and $(df(\bar{\mathbf{y}}))^{-1} = dh(\bar{\mathbf{x}})$. Hence, we have

$$\mathbf{Y}_i \mathbf{H} = dh(\bar{\mathbf{x}}) \mathbf{X}_i \mathbf{H} = \mathbf{L} \boldsymbol{\Sigma} \mathbf{V}', \quad \mathbf{L} \stackrel{\text{def}}{=} dh(\bar{\mathbf{x}}) \mathbf{U}, \quad (11.12)$$

which yields

$$\mathbf{L} = \mathbf{Y}_i \mathbf{H} \mathbf{V} \boldsymbol{\Sigma}^{-1}$$

and

$$\mathbf{Y}_i \mathbf{H} (\mathbf{I} - \mathbf{V} \mathbf{V}') = \mathbf{L} \boldsymbol{\Sigma} \mathbf{V}' - \mathbf{L} \boldsymbol{\Sigma} \mathbf{V}' \mathbf{V} \mathbf{V}' = 0. \quad (11.13)$$

11.1.3 Global Alignment

To make the global alignment for each local tangent approximation, we first place some constraints on the dimensionality reduction data matrix \mathbf{Y} . We require that \mathbf{Y} has zero mean and $\mathbf{Y}\mathbf{Y}' = \mathbf{I}$ (See Chapter 4 for the detailed discussion).

The key of the global alignment is to convert the local relation (11.13) to a global one. Let \mathbf{V} be the matrix in (11.9). We write $\mathbf{W}_i = \mathbf{H}(\mathbf{I} - \mathbf{VV}'')$. By (11.13),

$$\mathbf{Y}_i \mathbf{W}_i = \mathbf{0}.$$

We now extend the $d \times d$ matrix \mathbf{W}_i to an $n \times n$ matrix \mathbf{W}^i , which is defined by

$$\mathbf{W}^i(j, k) = \begin{cases} \mathbf{W}_i(s, l), & \text{if } j = i_s, k = i_l \in N(i), \\ \mathbf{0}, & \text{otherwise.} \end{cases}$$

In the similar way, we extend the matrix \mathbf{Y}_i to a $k \times n$ matrix \mathbf{Y}^i such that \mathbf{Y}_i is the submatrix of \mathbf{Y}^i with the column indices of $N(i)$, and other columns of \mathbf{Y}^i are zero. The equation (11.13) immediately yields

$$\mathbf{Y}^i \mathbf{W}^i = \mathbf{0}. \quad (11.14)$$

Because non-vanished columns of \mathbf{W}^i must have the indices in $N(i)$, we have

$$\mathbf{Y} \mathbf{W}^i = \mathbf{0}. \quad (11.15)$$

Furthermore, by $\mathbf{1}' \mathbf{H} = 0$, we also have

$$\mathbf{1}' \mathbf{W}^i = \mathbf{0}. \quad (11.16)$$

Then the data matrix \mathbf{Y} satisfies (11.13) for all $i = 1, 2, \dots, n$. Hence, we obtain the LTSA kernel

$$\mathbf{K} = \sum_{i=1}^n \mathbf{W}^i. \quad (11.17)$$

By (11.15) and (11.17), we have

$$\mathbf{Y} \mathbf{K} = \sum_{i=1}^n \mathbf{Y} \mathbf{W}^i = \mathbf{0}. \quad (11.18)$$

Similarly, by (11.16), we have

$$\frac{1}{n} \mathbf{1}' \mathbf{K} = \mathbf{0}. \quad (11.19)$$

Since \mathbf{K} is symmetric, by (11.18) and (11.19), we obtain

$$\mathbf{K} \left[\frac{1}{n} \mathbf{1} \mid \mathbf{Y}' \right] = \mathbf{0},$$

which indicates that the columns of the matrix $\begin{bmatrix} \frac{1}{n}\mathbf{1} & \mathbf{Y}' \end{bmatrix}$ form an o.n. basis of the null space of the kernel \mathbf{K} .

Finally, we point out that LTSA kernel \mathbf{K} is positive semi-definite. To prove it, we consider the matrix \mathbf{W}^i . It is clear that \mathbf{W}^i is positive semi-definite if and only if \mathbf{W}_i is positive semi-definite. By (11.16) and $\mathbf{P}_i = \mathbf{P}_i\mathbf{H}$, we have

$$\mathbf{U}\Sigma\mathbf{V}' = \mathbf{U}\Sigma\mathbf{V}'\mathbf{H},$$

which yields $\mathbf{V}' = \mathbf{V}'\mathbf{H}$ and therefore,

$$\mathbf{W}_i = \mathbf{H}(\mathbf{I} - \mathbf{V}\mathbf{V}') = \mathbf{H} - \mathbf{H}\mathbf{V}\mathbf{V}'\mathbf{H} = \mathbf{H}\mathbf{I}\mathbf{H} - \mathbf{H}\mathbf{V}\mathbf{V}'\mathbf{H} = \mathbf{H}(\mathbf{I} - \mathbf{V}\mathbf{V}')\mathbf{H}.$$

Since $\mathbf{V}\mathbf{V}'$ is a projection matrix, $\|\mathbf{V}\mathbf{V}'\| \leq 1$. Therefore, \mathbf{W}_i is positive semidefinite. Since \mathbf{K} is the sum of positive semidefinite matrices, $\mathbf{K} = \sum_{i=1}^n \mathbf{W}^i$, it is positive semidefinite too.

Considering possible errors of the approximation and the data deviation as well, we compute the DR data set \mathcal{Y} as the solution of the minimization problem

$$\mathbf{Y} = \underset{\mathbf{Y} \in \mathfrak{D}_{d,n}}{\arg \min} \text{tr}(\mathbf{Y}\mathbf{K}\mathbf{Y}'), \quad \text{s.t. } E(\mathbf{Y}) = 0. \quad (11.20)$$

Thus, the columns of the matrix \mathbf{Y}' are the eigenvectors of \mathbf{K} corresponding to the 2nd – $(d+1)$ st smallest eigenvalues.

11.2 LTSA Algorithm

11.2.1 LTSA Algorithm Description

According to the discussion of the previous section, an LTSA dimensionality reduction algorithm consists of the following steps.

Step 1. Neighborhood definition. The neighborhood of each point can be either k -neighborhood or ε -neighborhood. The number of points in a neighborhood is chosen to be greater than d . We denote by $G = [\mathcal{X}, \mathbf{A}]$ the data graph. Because the value ε in ε -neighborhood definition depends on the dimension and the scale of the data, k -neighborhood is more often used in neighborhood definition in LTSA algorithm.

Step 2. Local coordinate relation computation. Let $\mathbf{X}_i = [\mathbf{x}_{i_1} \dots \mathbf{x}_{i_k}]$ be the matrix whose columns are the k neighbors of \mathbf{x}_i . We centralize the data by subtracting their mean: $\hat{\mathbf{X}}_i = [\mathbf{x}_{i_1} - \bar{\mathbf{x}} \dots \mathbf{x}_{i_k} - \bar{\mathbf{x}}]$. The local coordinates (in \mathbb{R}^d) of $\hat{\mathbf{X}}_i$ is found by PCA: Let the d -rank

best approximation of $\hat{\mathbf{X}}_i$ be

$$\sum_{j=1}^d \sigma_j \mathbf{u}_j (\mathbf{v}_j)'.$$

Write $\mathbf{V} = [\mathbf{v}_1 \dots \mathbf{v}_d]$ and set $\mathbf{G}_i = \left[\frac{1}{k} \mathbf{1} \mid \mathbf{V}_i \right] \in \mathfrak{O}_{k,d+1}$. Then $\mathbf{W}_i = \mathbf{I} - \mathbf{G}_i \mathbf{G}'_i$.

Step 3. LTSA kernel construction via global alignment. The LSTA kernel \mathbf{K} is the alignment matrix of all local matrices \mathbf{W}_i . Initialize \mathbf{K} by a zero matrix. Let $N(i)$ be the index set of the neighborhood of \mathbf{x}_i and $\mathbf{K}(N(i), N(i))$ be the submatrix of \mathbf{K} containing the rows and columns of the indices $N(i)$. Then update \mathbf{K} by setting $\mathbf{K}(N(i), N(i)) = \mathbf{I} - \mathbf{G}_i \mathbf{G}'_i$ for $i = 1, 2, \dots, n$, successively.

Step 4. Eigen decomposition of DR kernel. Let the spectral decomposition of \mathbf{K} be given by

$$\mathbf{K} = \mathbf{U} \Lambda \mathbf{U}',$$

where $\Lambda = \text{diag}(\lambda_0, \lambda_1, \dots, \lambda_{n-1})$ with

$$0 = \lambda_0 \leq \lambda_1 \leq \dots \leq \lambda_{n-1}.$$

Assume that the graph $G = [\mathcal{X}, \mathbf{A}]$ is connected, then $\lambda_1 > 0$. Let $\mathbf{Y} = [\mathbf{u}_1 \dots \mathbf{u}_d]'$ be the transpose of the eigenvector matrix corresponding to the 2nd – $(d+1)$ st smallest eigenvalues of \mathbf{K} . Then \mathcal{Y} is the DR data set. This step is the same as in LLE algorithm.

11.2.2 Matlab Code of LTSA

The MATLAB code of LTSA algorithm is the following.

```
function [Y, eigv] = drltsa(X, d, options)
% Locally tangent planes alignment for DR
% SYNTAX:
%   Y = drltsa(X, d, options);
% INPUTS:
% X: D x n data matrix, X(:,i) is i-th point.
% d: Dimension of output data.
% OPTIONS:
%   .epsilon: epsilon-neighbors.
%   .k: k-neighbors.
%   .verb: display the processing verbally.
% OUTPUT:
%   Y: d x n dimension-reduced data matrix.
%   eigv: 1 - eigenvalues.
% Example: Reduce Swiss roll to 2-D.
```

```

% N=1000;
% tt = (3*pi/2)*(1+2*rand(1,N));
% height = 21*rand(1,N);
% X = [tt.*cos(tt); height; tt.*sin(tt)];
% d=2;
% options.k=10;
% [Y,ev] = DRLTSA(X, d, options);
%
% CALL: data_neighborhood
%
% LTSA code is originally written
% by Zhenyue Zhang & Hongyuan Zha (2004) in
% http://pubs.siam.org/sam-bin/dbq/article/41915
%
% This code is modified from original code
% by Jianzhong Wang, 2008.
%
% Initialize inputs.
options.null=0;
n = size(X, 2);
if ~isfield(options, 'epsilon')
    if isfield(options, 'k')
        k = options.k;
        if numel(k)==1
            k = k*ones(n,1);
        end
    else
        k = 10*ones(n,1);
    end
end
if isfield(options, 'verb')
    verb = option.verb;
else
    verb = 1;
end
% initialize kernel.
B = sparse(n,n);
% Step 1. Construct data-graph.
if verb
    fprintf(1,'--Construct data graph\n');
end
[D2, nghd] = data_neighborhood(X,options);
% find # of neighbors for epsilon-neighborhood.
if isfield(options, 'epsilon')
    k = sum(nghd,2);
end
if verb
    fprintf(1,'--Construct LTSA kernel\n');
end
for i = 1:n

```

```

% Step 2. Construct local LTSA kernel
%take k nearest neighbors.
thisi = find(nghd(:,i));
thisx = X(:,thisi);
z = thisx-repmat(mean(thisx,2),1,k(i));
[Vi,cv] = svd(z',0);
Gi = [repmat(1/sqrt(k(i)),k(i),1) Vi(:,1:d)];
% Step 3. Construct LTSA kernel
% by global alignment.
BII = eye(k(i))-Gi*Gi';
B(thisi,thisi) = B(thisi,thisi)+BII;
end
% force B to be symmetric.
B = (B+B')/2;
% Step 4. Eigen decomposition.
if verb
    fprintf(1,'--Find dimension-reduced data.\n');
end
options.disp = 0;
options.isreal = 1;
options.issym = 1;
[Y, eigv] = eigs(B,d+1,'SM',options);
% bottom evect is [1,1,1,...] with eval 0.
% Scale the output.
Y = Y(:,1:d)'*sqrt(n);

```

11.3 Experiments of LTSA Algorithm

11.3.1 Test LTSA on Artificial Surfaces

In Fig. 11.2, LTSA algorithm is applied to reduce the data of 4 artificial surfaces, 3D-cluster, Punched sphere, Swiss roll, and S-curve to 2-dimensional data, where 10 nearest points are used to construct the neighborhood. The experiment shows that LTAS works well for these surfaces, except 3D-cluster. Note that the data set of 3D-cluster lacks the surface structure on the line segments. Compared to LLE, LTSA does the better job on Swiss roll.

Noise on data impacts the dimensionality reduction results very much. Because LTSA adopts the local approximation using tangent space, it is more sensitive to noise. When we apply LTSA algorithm to noise data with the standard deviation 0.2 and 0.4, we cannot obtain the valid DR data for these surfaces. When we reduce the noise level to 0.1, the DR data are valid for Swiss roll and S-curve only. In Fig. 11.3, the two valid DR results are displayed.

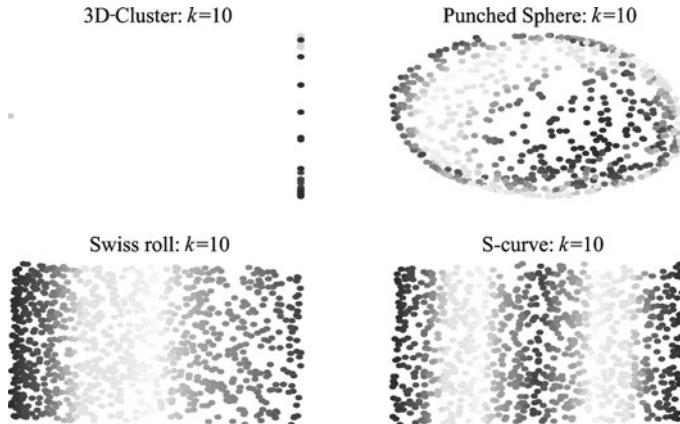


Fig. 11.2 LTSA algorithm is tested on 4 surfaces: 3D-cluster, Punched sphere, Swiss roll, and S-curve. 1000 points are randomly sampled on each surface and 10-neighborhood is chosen to construct the data graphs. All of these surfaces are reduced to 2-dimensional data (on the plane). The figure displays the DR results of these surfaces. The original 3-dimensional graphs are not displayed.

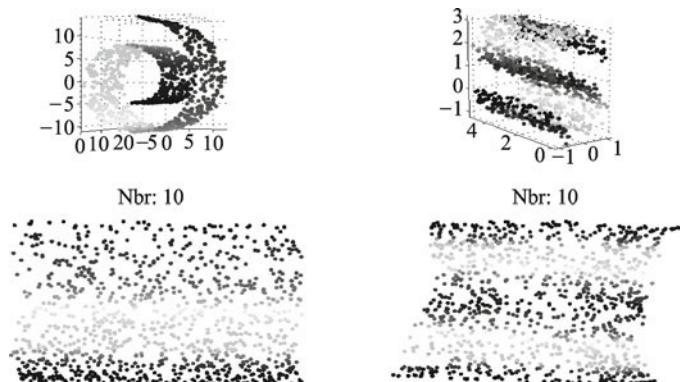


Fig. 11.3 LTSA algorithm is applied on 2 surfaces with the noise standard deviation 0.1. The algorithm is relatively sensitive to noise. When the noise level is high, we cannot obtain valid DR results. Even though the noise standard deviation is already reduced to 0.1, the algorithm can only produce the valid results for Swiss roll and S-curve. The settings for the test are the same as above: 1000 data points are randomly sampled on each surface and 10-neighborhood is used for the construction of data graphs. Top left: Swiss roll. Top right: S-curve. Their corresponding DR data sets are displayed on the bottom line.

LTSA algorithm is relatively sensitive to the neighborhood sizes, only stable over a relatively narrow range. The range depends on various features of the data, such as the sampling density and the manifold geometry. The experiment shows that when we perform LTSA for the above 4 surfaces, it is

unstable if the neighborhood size is less than 7. It is also shown that when the size is larger than 12, the DR data distorts the local geometry of the original data. The results are displayed in Fig. 11.4.

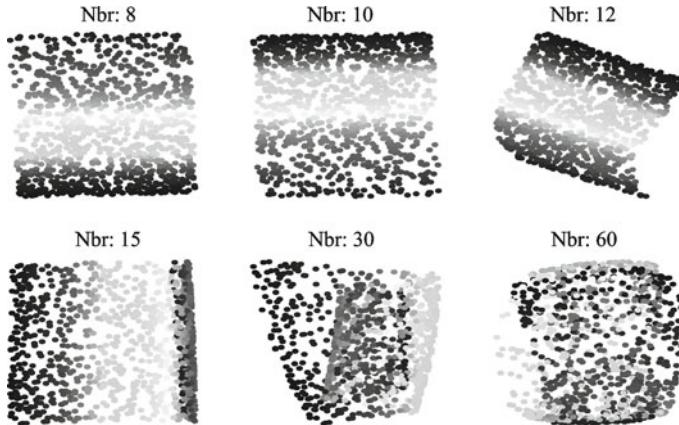


Fig. 11.4 Effect of neighborhood size on LTSA. LTSA is quite sensitive to the neighborhood size. In this figure, LTSA is applied to Swiss roll for different choices of the number of nearest neighbors K , 8, 10, 12, 15, 30, 60, respectively. When K is chosen to be less than 7, the algorithm becomes unstable and no DR result is produced. Unlike LLE, the reliable LTSA embedding from 3 dimension to 2 dimension can only be obtained in a narrow range of the sizes. If K is too small (less than 7), then no valid results can be produced. If K is a little larger, say, larger than 14, LTSA does not succeed in preserving data geometry.

The shape of the surface also impacts the results of LTSA dimensionality reduction. In Fig. 11.5, LTSA algorithm is applied to Swiss rolls with the lengths 21, 51, and 81, respectively. The number of nearest points $K = 10$ is chosen for the experiment. The result is not very satisfied when the length of the roll is too long or too short.

In Fig. 11.6 LTSA algorithm is applied to Swiss rolls with the lengths 21, 51, and 81, respectively. The results show that when the length is 81, the DR result is not valid. This phenomenon can be explained as follows. When the length of the roll is 81 while the sample points keep the same number of 1000 as for the short roll of length 21, the distances between the points in a neighborhood for the long roll are four times of those for the short roll in average. Therefore, the graph neighborhood of the long roll may not reside on the manifold neighborhood so that the DR result cannot preserve the geometric structure of the manifold. In Fig. 11.7, the different lengths of S-curve, 5, 10, 30, are chosen for the test with the same parameter settings as in Fig. 11.2. By the same argument we have made for the Swiss rolls in Fig. 11.6, the distortion of the DR data for the long S-curve is perhaps due to the neighborhood inconsistence.

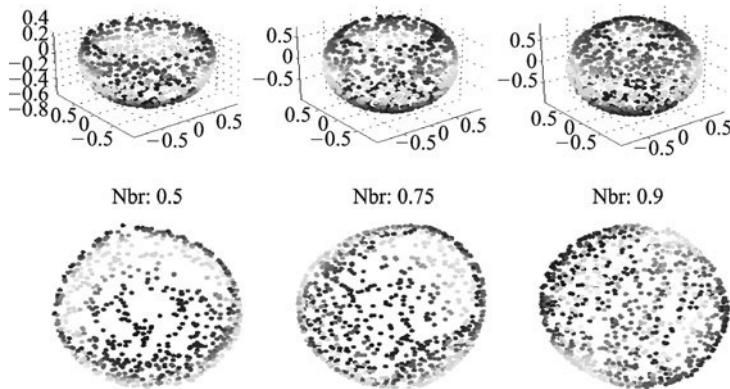


Fig. 11.5 LTSA algorithm is performed on the punched spheres with various heights. On the top line, we present three punched spheres with heights 0.5, 0.75, and 0.9 respectively. The diameter of the sphere is 1. On the bottom line are their DR data sets correspondingly. The results show that the DR data sets do not preserve the local geometry near the boundary when the punched holes become smaller.

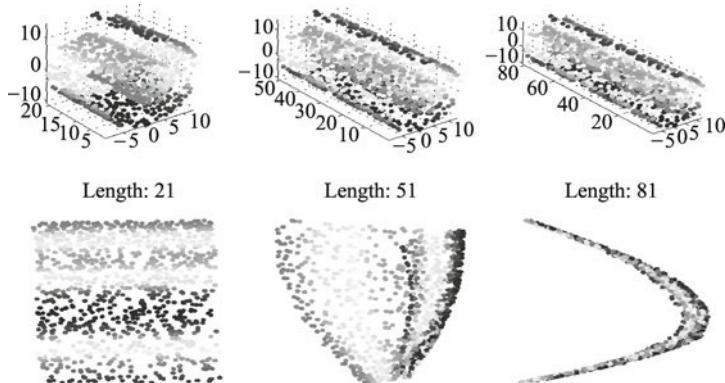


Fig. 11.6 Effect of surface size on LTSA. LTSA embeddings of Swiss-Roll are computed for different choices of the lengths of the roll, 21, 51, 81, respectively. It is shown that the dimensionality reduction for the Swiss roll with length 51 is greater than other two.

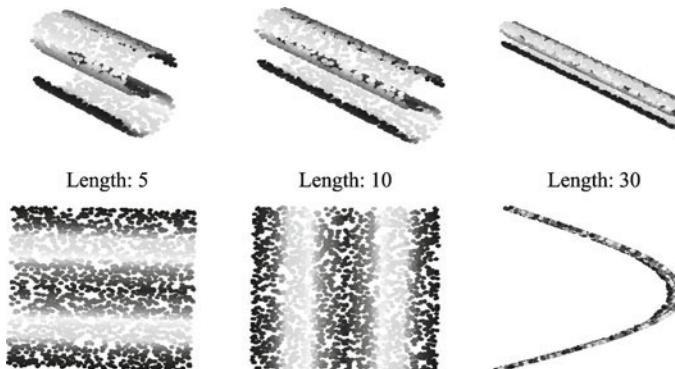


Fig. 11.7 The figure illustrates again that when the graph neighborhood is inconsistent with manifold, the DR data may distort the geometric structure so that it becomes invalid. In this test, LTSA embeddings of the S-curves are computed for different choices of the lengths, 5, 10, and 30, respectively. The surface data are sampled in the same way as in Fig. 11.2. The results show that the dimensionality reduction becomes worse as the length increases, due to the neighborhood inconsistency.

11.3.2 Conclusion

Similar to LLE, the LTSA kernel is sparse, and hence LTSA is a fast algorithm. Because LTSA uses tangent spaces to approximate the data, it usually produces satisfactory DR results for the data sampled on smooth and connected manifolds. For non-smooth surfaces, such as 3D-cluster, it cannot produce the valid DR result. Furthermore, since LTSA is quite sensitive to the smoothness, when the data are contaminated with noise, the algorithm becomes unstable. The algorithm is also relatively sensitive to the neighborhood size. Hence, when LTSA algorithm is adopted, the neighborhood size has to be carefully chosen.

References

- [1] Zhang, Z.Y., Zha, H.Y.: Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. *SIAM J. Sci. Comput.* 26(1), 313–338 (2004).
- [2] Lin, T., Zha, H.B.: Riemannian manifold learning. *IEEE Trans. on Pattern Anal. and Machine Intel.* 30(5), 796–809 (2008). <http://dx.doi.org/10.1109/TPAMI.2007.70735>. Accessed 1 December 2011.
- [3] Lin, T., Zha, H.B., Lee, S.U.: Riemannian manifold learning for nonlinear dimensionality reduction. In: *ECCV I*, pp. 44–55 (2006). http://dx.doi.org/10.1007/11744023_4. Accessed 1 December 2011.

- [4] Zhang, Z.Y., Zha, H.Y., Zhang, M.: Spectral methods for semi-supervised manifold learning. In: CVPR, pp. 1–6 (2008).
- [5] Zhao, D.: Formulating LLE using alignment technique. Pattern Recognition 39(11), 2233–2235 (2006).

Chapter 12 Laplacian Eigenmaps

Abstract Laplacian eigenmaps (Leigs) method is based on the idea of manifold unsupervised learning. Let \mathcal{X} be the observed high-dimensional data, which reside on a low-dimensional manifold M . Let h be the coordinate mapping on M so that $\mathcal{Y} = h(\mathcal{X})$ is a DR of \mathcal{X} . Each component of the coordinate mapping h is a linear function on M . Hence, all components of h nearly reside on the numerically null space of the Laplace-Beltrami operator on M . In Leigs method, a Laplace-Beltrami operator is constructed on the data graph. Then the DR data set is derived from the eigenvectors of the operator corresponding to several smallest eigenvalues. The chapter is organized as follows. In Section 12.1, we describe the Laplacian eigenmaps method and focus the discussion on the construction of the Laplace-Beltrami operator on the data set. In Section 12.2, the Leigs DR algorithm is developed. In Section 12.3, some experiments of the algorithm are presented.

12.1 Description of the Laplacian Eigenmap Method

The Laplacian eigenmaps method (Leigs) was introduced by Belkin and Niyogi [1, 2], using the idea of the manifold learning [3–5]. While the idea of Laplacian eigenmaps method is similar to that of LLE, its DR kernel is derived from the Laplace-Beltrami operator on the manifold, where the data set resides. To describe the method, we adopt the same data model as before. Let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ be the data set, which resides on an underlying d -dimensional manifold $M \subset \mathbb{R}^D$. Let g be a parametrization of M and $h = g^{-1} = [h^1, \dots, h^d]'$ be the coordinate mapping on M such that for each $\mathbf{x} \in M$, $\mathbf{y} = h(\mathbf{x}) \in \mathbb{R}^d$ provides d coordinates for \mathbf{x} . Then the data set

$$\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\}, \quad \mathbf{y}_i = h(\mathbf{x}_i), 1 \leq i \leq n \quad (12.1)$$

is the DR set. To find \mathcal{Y} , we have to learn the coordinate mapping from the observed data set \mathcal{X} , assuming that a neighborhood system is given on \mathcal{X} . Let $G = [\mathcal{X}, A]$ be the graph on \mathcal{X} , which defines the neighborhood

system. We assume that the graph G is undirected, i.e., \mathbf{A} is symmetric, and the neighborhood system on \mathcal{X} is consistent with the geometry on M . Let O_i be the graph neighborhood of \mathbf{x}_i . Since the neighborhood is well defined, each point $\mathbf{x}_j \in O_i$ is near \mathbf{x}_i on M .

The idea of Laplacian eigenmaps method can be simply explained as follows. Let $C^2(M)$ be the space of twice differentiable functions on M , and \mathcal{L} be the Laplace-Beltrami operator on $C^2(M)$, which is defined by (see Section 2.3)

$$\mathcal{L}u = -\operatorname{div}(\operatorname{grad}(u)), \quad u \in C^2(M).$$

Let e denote the constant function on M with $e(\mathbf{x}) = 1, \mathbf{x} \in M$. Then e is in the null space of the Laplace-Beltrami operator \mathcal{L} . Therefore, it can be obtained as the solution of the equation

$$\mathcal{L}f = 0. \quad (12.2)$$

Since each coordinate function h^l is linear on M , the totality of $h^l, 1 \leq l \leq d$, can be approximately represented by the linear combinations of achieving the d smallest positive eigenvalues. They are called feature functions. In Leigs method, the DR data are derived from these feature functions instead of the coordinate ones.

In general, data sets may be contaminated by noise, and there are many other facts which may disturb the solutions. Hence, the equation model (12.2) needs to be slightly modified. The relation (see Section 2.3)

$$\int_M f \mathcal{L}(f) = \int_M \|\nabla f\|^2 \quad (12.3)$$

inspires us to modify the equation model (12.2) to the following minimization model

$$h = \arg \min \int_M f \mathcal{L}(f) \quad (12.4)$$

under certain constraints. Leigs method adopts the model (12.4) to find the DR set \mathcal{Y} .

12.1.1 Approximation of Laplace-Beltrami Operator

The key step of Leigs method is to construct the Laplace-Beltrami operator \mathcal{L} . By (12.3), the Laplace-Beltrami operator \mathcal{L} is positive and self-adjoint. Applying the exponential formula to positive and self-adjoint operators [6], we can construct a single-parametric semi-group

$$\mathcal{A}_t = \exp(-t\mathcal{L}), \quad (12.5)$$

where \mathcal{A}_t is called the *Neumann heat diffusion operator* on M having the following integration expression.

$$\mathcal{A}_t(f)(\mathbf{x}) = \int_M G_t(\mathbf{x}, \mathbf{y})f(\mathbf{y}), \quad f \in C(M). \quad (12.6)$$

In (12.6), the integral kernel G_t is known as the Neumann heat kernel, which approximates the Gaussian [7]:

$$G_t(\mathbf{x}, \mathbf{y}) = (4\pi t)^{-d/2} e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{4t}} (\phi(\mathbf{x}, \mathbf{y}) + O(1)), \quad (12.7)$$

where ϕ is a smooth function with $\phi(\mathbf{x}, \mathbf{x}) = 1$ and $(4\pi t)^{d/2}$ is the normalization factor computed from

$$(4\pi t)^{d/2} \approx \int_M e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{4t}}. \quad (12.8)$$

Hence, we have [7]

$$G_t(\mathbf{x}, \mathbf{y}) \approx (4\pi t)^{-d/2} e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{4t}}. \quad (12.9)$$

By the exponential formula (12.5),

$$\mathcal{L} = s - \lim_{t \rightarrow 0^+} \frac{I - \mathcal{A}_t}{t}. \quad (12.10)$$

When t is close to 0, (12.10) yields the following approximation.

$$\mathcal{L}f(\mathbf{x}) \approx \frac{1}{t} \left(f(\mathbf{x}) - (4\pi t)^{-d/2} \int_M e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{4t}} f(\mathbf{y}) \right). \quad (12.11)$$

12.1.2 Discrete form of Laplace-Beltrami Operator

To construct the Leigs DR kernel, we need to discretize the expression of the continuous Laplace-Beltrami operator in (12.11). A function f on \mathcal{X} can be represented as a vector $\mathbf{f} = [f(\mathbf{x}_1), \dots, f(\mathbf{x}_n)]'$. Intuitively, we might discretize (12.11) to

$$\mathcal{L}f(\mathbf{x}_i) = \frac{1}{t} \left(f(\mathbf{x}_i) - (4\pi t)^{-d/2} \sum_{\mathbf{x}_j \in O_i} e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{4t}} f(\mathbf{x}_j) \right). \quad (12.12)$$

Note that the normalization factor $(4\pi t)^{d/2}$ should make

$$(4\pi t)^{-d/2} \sum_{\mathbf{x}_j \in O_i} e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{4t}} = 1.$$

Unfortunately, it is not true in the discrete case. To ensure that the above sum is properly normalized to have the sum 1, we have to replace $(4\pi t)^{d/2}$ by

$$d_i = \sum_{\mathbf{x}_j \in O_i} e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{4t}}.$$

However, the replacement above destroys the self-adjoint property of \mathcal{L} . Hence, to break away from the dilemma, as discussed in Section 3.3, we first define the (non-normalized) discrete Laplacian \mathbf{L} by

$$\mathbf{L}f(\mathbf{x}_i) = \frac{1}{t} \left(d_i f(\mathbf{x}_i) - \sum_{\mathbf{x}_j \in O_i} e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{4t}} f(\mathbf{x}_j) \right), \quad (12.13)$$

which can be shortly rewritten as

$$t\mathbf{L}f(\mathbf{x}_i) = d_i f(\mathbf{x}_i) - \sum_{j=1}^n w_{i,j} f(\mathbf{x}_j), \quad (12.14)$$

with

$$w_{i,j} = \begin{cases} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{4t}\right), & \text{if } \mathbf{x}_j \in O_i, \\ 0, & \text{otherwise.} \end{cases} \quad (12.15)$$

Let

$$\mathbf{D} = \text{diag}(d_1, \dots, d_n), \quad \mathbf{W} = [w_{i,j}]_{i,j=1}^n. \quad (12.16)$$

Then the matrix form of the non-normalized discrete Laplacian \mathbf{L} is

$$\mathbf{L}(\mathbf{f}) = (\mathbf{D} - \mathbf{W})\mathbf{f},$$

where t is removed from the equation since it does not impact the solution of the problem. In Section 3.3, we already proved that Laplacian \mathbf{L} is positive and self-adjoint. It is also known that the discrete Laplace-Beltrami operator \mathcal{L} can be derived from \mathbf{L} by the formula

$$\mathcal{L} = \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}. \quad (12.17)$$

12.1.3 Minimization Model for DR Data Set

We return to the discussion on the minimization problem for the DR data set $\mathcal{Y} = h(\mathcal{X})$. Write $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_n] \subset \mathbb{R}^d$ and recall that the i th row of \mathbf{Y} , denoted by \mathbf{Y}_i , is the i th feature function on the set \mathcal{X} :

$$\mathbf{Y}_i = [h^i(\mathbf{x}_1), \dots, h^i(\mathbf{x}_n)].$$

As discussed in Chapter 4, \mathbf{Y} is required to be centered and have o.g. rows:

$$\mathbf{Y}\mathbf{1} = 0, \quad \mathbf{Y}\mathbf{Y}' = \mathbf{I}. \quad (12.18)$$

By (12.4), we set \mathbf{Y} to be the solution of the following discretized minimization problem:

$$\mathbf{Y} = \arg \min \text{tr}(\mathbf{Y}\mathcal{L}\mathbf{Y}'), \quad (12.19)$$

subject to the constraints (12.18).

The problem (12.19) can be solved by the eigen decomposition of \mathcal{L} . We assume that the graph $G = [\mathcal{X}, \mathbf{A}]$ is connected. Hence, 0 is the simple eigenvalue of \mathcal{L} and $\mathbf{D}^{1/2}\mathbf{1}$ is the 0-eigenvector. Let all eigenvalues of \mathcal{L} be arranged in the order of

$$0 = \lambda_0 < \lambda_1 \leq \lambda_2 \cdots \leq \lambda_{n-1}.$$

Then \mathbf{Y}'_i is the eigenvector of \mathcal{L} corresponding to λ_i , $1 \leq i \leq d$.

Recall that the eigen problem

$$\mathcal{L}\mathbf{a} = \lambda\mathbf{a} \quad (12.20)$$

is equivalent to the generalized eigen problem

$$\mathbf{L}\mathbf{b} = \lambda\mathbf{D}\mathbf{b} \quad (12.21)$$

with the relation $\mathbf{a} = \mathbf{D}^{1/2}\mathbf{b}$. Since \mathbf{Y}'_i is the solution of (12.20) so that

$$\mathcal{L}\mathbf{Y}'_i = \lambda_i\mathbf{Y}'_i, \quad 1 \leq i \leq d,$$

we conclude that $\mathbf{B}'_i = \mathbf{D}^{-1/2}\mathbf{Y}'_i$ is the solution of (12.21):

$$\mathbf{L}\mathbf{B}'_i = \lambda_i\mathbf{D}\mathbf{B}'_i.$$

Hence, alternatively, we can first solve (12.21) to obtain

$$\mathbf{B}' = [\mathbf{B}'_1, \dots, \mathbf{B}'_d],$$

and then obtain the DR data matrix by

$$\mathbf{Y} = \mathbf{B}\mathbf{D}^{1/2}.$$

Remark 12.1. If we ignore the data dilation caused by $\mathbf{D}^{1/2}$, we can identify \mathbf{Y} with \mathbf{B} . Hence, in Leigs DR algorithm, we sometimes set \mathbf{B} as the output. A reason to use Equation (12.21) to replace (12.20) is that the computation of the kernel \mathbf{L} saves the normalization step, that may accelerate the computing speed. Therefore, \mathbf{L} is often called *Leigs kernel* or *graph Laplacian*.

12.1.4 Construction of General Leigs Kernels

As shown in the previous subsection, the DR data set \mathbf{B} is the solution of the minimization problem

$$\mathbf{B} = \arg \min \text{tr}(\mathbf{B}(\mathbf{D} - \mathbf{W})\mathbf{B}'), \quad \text{s.t. } \mathbf{B}\mathbf{D}\mathbf{B}' = \mathbf{I}, \quad \mathbf{B}\mathbf{D}\mathbf{1} = \mathbf{0}, \quad (12.22)$$

where

$$\text{tr}(\mathbf{B}(\mathbf{D} - \mathbf{W})\mathbf{B}') = \sum_{i=1}^n d_i \mathbf{b}_i - \sum_{i,j=1}^n w_{i,j} \langle \mathbf{b}_i, \mathbf{b}_j \rangle \quad (12.23)$$

$$= \frac{1}{2} \sum_{i,j=1}^n w_{i,j} \|\mathbf{b}_i - \mathbf{b}_j\|^2, \quad (12.24)$$

and $w_{i,j}$ is defined by (12.15). The objective function above with the choice of weights $w_{i,j}$ incurs a heavy penalty if neighboring points \mathbf{x}_i and \mathbf{x}_j are far apart from each other. Therefore, minimizing it is an attempt to ensure that if \mathbf{x}_i and \mathbf{x}_j are close, then \mathbf{b}_i and \mathbf{b}_j are close too. By the interpretation above, the weight matrix \mathbf{W} in the minimization problem (12.22) is not necessarily derived from the Gaussian. Generally, let $g > 0$ be a decreasing function on $[0, \infty)$. Then the weight matrix defined by

$$w_{i,j} = \begin{cases} g(\|\mathbf{x}_i - \mathbf{x}_j\|^2), & \text{if } \mathbf{x}_j \in O_i, \\ 0, & \text{otherwise} \end{cases} \quad (12.25)$$

can be used in the model (12.19). When a general function g is applied in the construction of \mathbf{W} , we shall call (12.22) a *generalized Leigs model*. This generalization widens the contents of Leigs method. As an important extension, we may directly adopt the adjacency matrix \mathbf{A} of the data graph as the weight matrix \mathbf{W} in (12.22). This design of \mathbf{W} can also be considered as setting $t = \infty$ in (12.15).

12.2 Laplacian Eigenmaps Algorithm

12.2.1 Steps in Leigs Algorithm

Leigs algorithm consists of the following steps.

Step 1. Neighborhood definition. This step is similar to other nonlinear DR algorithms. Either k -neighborhood or ε -neighborhood can be used for the construction of graph on the data \mathcal{X} . In applications, k -neighborhood is more common than ε -neighborhood. To ensure that the graph is undirected, when k -neighborhood is applied, a symmetric modification is needed for obtaining a symmetric adjacency matrix \mathbf{A} .

Step 2. Weighted graph creation. The Leigs algorithm can simply adopt the **symmetric adjacency \mathbf{A} as the weight matrix**. It can also use the formula (12.15), even the more general formula (12.25), to construct \mathbf{W} . Using \mathbf{A} as the weight matrix has the advantage that the constructed

weight matrix has no parameter. Otherwise, say, (12.15) is adopted, we have a parameter t in the determination of the weights. Implementation shows that the Leigs DR algorithm is not very sensitive to t in general. Hence, it is quite common using parameter-free weight matrix in Leigs algorithm.

Step 3. DR kernel construction. Let the weight matrix be denoted by $\mathbf{W} = [w_{i,j}]$. let $d_i = \sum_j w_{i,j}$ and $\mathbf{D} = \text{diag}(d_1, \dots, d_n)$. Then $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is the Leigs DR kernel.

Step 4. Eigen decomposition of DR kernel. We solve the generalized eigen problem:

$$\mathbf{L}\mathbf{f} = \lambda \mathbf{D}\mathbf{f}. \quad (12.26)$$

Let $\{\lambda_i\}_{i=0}^{n-1}$, with $0 = \lambda_0 \leq \lambda_1 \leq \dots \leq \lambda_{n-1}$, be the eigenvalues of (12.26), and $\mathbf{F} = [\mathbf{f}^1 \ \dots \ \mathbf{f}^d]$ be the harmonic eigenvector matrix corresponding to the eigenvalue set $\{\lambda_1, \dots, \lambda_d\}$. Then $\mathbf{Y} = \mathbf{F}'$ is the required DR data matrix. If scaling is required, we set $\mathbf{Y} = (\mathbf{F}\mathbf{D}^{1/2})'$.

12.2.2 Matlab Code of Leigs Algorithm

The Matlab code of Leigs algorithm adopts the adjacency matrix as the weight matrix. The reader can easily modify the algorithm using the formula in (12.15) to create parametric weight matrix.

```
function [Y,V] = drleigs(X, d, options)
% Laplacian Eigenmaps Algorithm for DR
% SYNTAX:
%   Y = DRLEIGS(X,d,options)
% INPUTS:
%   X: D x n data matrix, X(:,i) is the i-th point.
%   d: Dimension of output data.
% OPTIONS:
%   .epsilon: epsilon-neighbors.
%   .k: k-neighbors.
%   .par: Use parametric weights model
%         with parameter par (= -4t).
%   .scalingY: Scale output Y.
%   .verb: display the processing verbally.
% OUTPUT:
%   Y: d x n dimension reduced data matrix.
%   V: 1 - eigenvalues corresponding to Y.
% Example: Reduce Swiss roll to 2-D.
%   N=1000;
%   tt = (3*pi/2)*(1+2*rand(1,N));
%   height = 21*rand(1,N);
%   X = [tt.*cos(tt); height; tt.*sin(tt)];
%   d=2;
```

```

%     options.k=10;
%     [Y,ev] = DRLEIGS(X, d, options);
%
% CALL: data_neighborhood
%
% Algorithm LEIGS refers to University of Chicago
% Computer Science Technical Report TR-2002-01
% M. Belkin and P. Niyogi misha@math.uchicago.edu
% http://www.cs.uchicago.edu/research/publications
%         /techreports/TR-2002-1
%
% Code is modified by the author
%
% Initialize options.
N = size(X,2);
options.symmetric = 1;
if ~isfield(options, 'scalingY')
    scalingY= false;
else
    scalingY= options.scalingY;
end
if ~isfield(options,'par')||isempty(options.par)
    parmode = false;
else
    parmode = true;
    par = options.par;
end
if isfield(options, 'verb')
    verb = option.verb;
else
    verb = 1;
end
% Step 1: Construct data-graph.
if verb
    fprintf(1,'--Construct data graph\n');
end
[D2, A] = data_neighborhood(X,options);
% Step 2: Construct LEIGS kernel
if verb
    fprintf(1,'--Construct LEIGS kernel.\n');
end
if ~parmode
    W = A;
else
    W = speye(N)+ spfun(@exp, -D2/par);
end
DW = sum(W,2);
D = spdiags(DW,0,speye(N));
L = D-W;
% Step 3: Eigen decomposition

```

```

if verb
    fprintf(1,'--Find dimension-reduced data.\n');
end
options.tol = 1e-9; options.disp = 0;
options.isreal = 1; options.issym = 1;
[Y,V] = eigs(L,D, d+1,'SM',options);
Y = Y(:,1:d);
if scalingY
    Y = sqrt(D)*Y;
end
Y=Y';
V = diag(V);

```

12.3 Implementation of Leigs Algorithm

12.3.1 Experiments on Artificial Surfaces

In this section, we demonstrate the Leigs method in the dimensionality reduction for artificial surfaces. For comparing Leigs method with other nonlinear DR methods, we adopt the same settings on the data for testing the algorithm. In Fig. 12.1, the 3-dimensional data of 4 artificial surfaces, 3D-cluster,

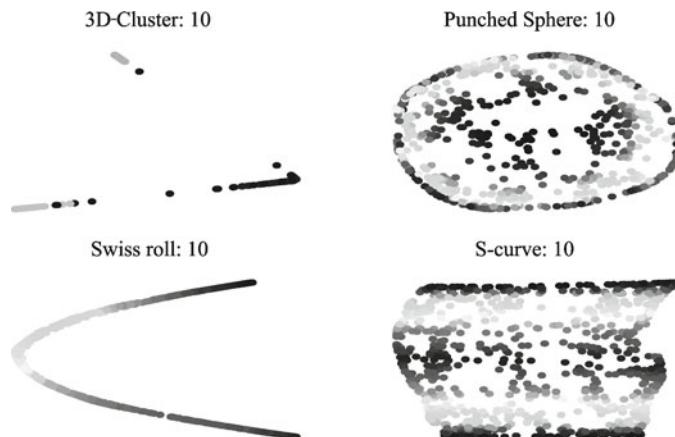


Fig. 12.1 Leigs algorithm is tested on 4 surfaces: 3D-cluster, Punched sphere, Swiss roll, and S-curve. 1000 points are randomly sampled on each of them. All are reduced to 2-dimensional data (on the plane). The original 3-dimensional graphs of these surfaces are not presented here. They can be found in Section 1.5. The figure displays the DR results for these surfaces. The graph shows that Leigs algorithm works well for S-curve and Punched sphere.

Punched sphere, Swiss roll, and S-curve are reduced to 2-dimensional ones by Leigs algorithm. In the experiment, 1000 points are randomly sampled on each surface and 10 nearest points are used to construct the neighborhood of a point. The experiment shows that Leigs algorithm works well for S-curve and Punched sphere. It cannot produce a valid DR for 3D-cluster and the DR data is not well scaled for Swiss roll. Note that Leigs is based on Laplace-Beltrami operator. Hence, it may not perform well in non-smooth area of a surface. But the data of 3D-cluster shows a lack of the local geometry in the area around the line segments. It is reasonable that Leigs does not produce a valid DR data set for 3D-cluster. The coordinate mapping in Leigs is not an isometric mapping. Hence, it usually does not well preserve the local Euclidean distances for the data such as Swiss roll. Noise on data impacts the dimensionality reduction results. In Fig. 12.2 and Fig. 12.3, we apply Leigs algorithm to reduce the dimension of noisy data, with the noise standard deviation 0.2 and 0.4 respectively. The results show that Leigs algorithm can tolerate noise with the deviation in a certain range, producing better DR results than LTSA and LLE. This is perhaps due to the diffusion structure of the Leigs DR kernel. Recall that the Leigs DR kernel is derived from a diffusion kernel and a diffusion processing reduces the noise impact. When strong noise exists, the DR data may not preserve the data geometry very well since the data have a large deviation from the underlying manifold.

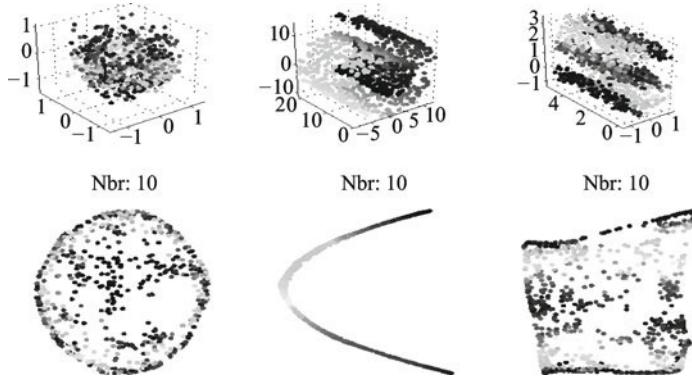


Fig. 12.2 Leigs algorithm is applied on three surfaces with the noise standard deviation 0.2. All of the surfaces are sampled in the same way as in Fig. 12.1. The figure shows that Leigs algorithm can tolerate noise in a certain range. At the top line, three noisy data are presented. Top left: Punched sphere, Top middle: Swiss roll. Top right: S-curve. At the bottom line are their corresponding dimensional reduced 2-dimensional data.

As we discussed before, the properties of the underlying manifold greatly impact the results of the DR. In Fig. 12.4, the different heights of Punched sphere are chosen for the test. It is not surprise that when the height increases, the DR results have more distortion. Compared with LTSA and LLE, Leigs has better performance. In Fig. 12.5 Leigs algorithm is applied to Swiss

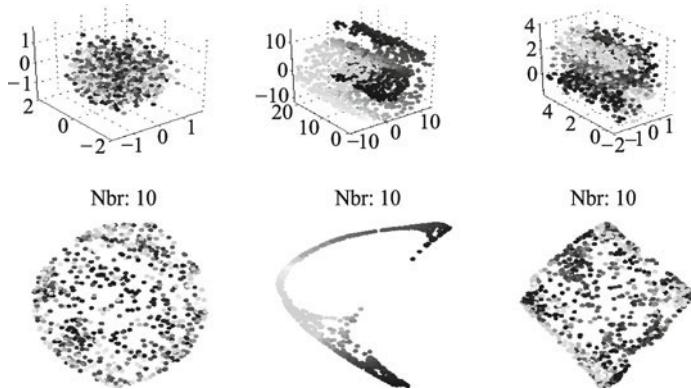


Fig. 12.3 Leigs algorithm is applied on three surfaces with the noise standard deviation 0.4. All of the surfaces are sampled in the same way as in Fig. 12.1. The figure shows that when the noise deviation is large, the DR results of Leigs algorithm have distortion at a certain level. At the top line, three noisy data are presented. Top left: Punched sphere, Top middle: Swiss roll. Top right: S-curve. At the bottom line are their corresponding dimensional reduced 2-dimensional data.

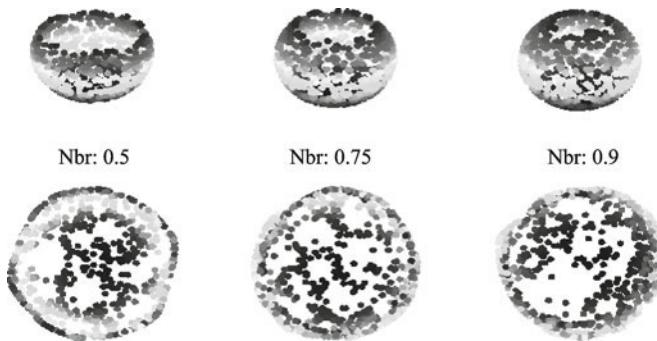


Fig. 12.4 Effect of surface shape on Leigs (I). Embeddings of the punched spheres are computed for different choices of the heights of the punched sphere, 0.5, 0.75, and 0.9, respectively, where the diameter of the original sphere is unit. It shows that the dimensionality reduction for the punched sphere becomes worse as the height increases.

rolls with different lengths 21, 51, and 81, respectively. The results show that when the length is 81, the DR result is not valid. This phenomenon can be explained as follows. When the length of the roll is 81, the surface area is four times of the roll with the length 21. Since the sample points keep the same number of 1000, the distances between the points in a neighborhood for the long roll is also four times of those for the short roll in average. Therefore, when the length is too large, the graph neighborhood does not truly reside on a neighborhood of the manifold, the DR result cannot preserve the geomet-

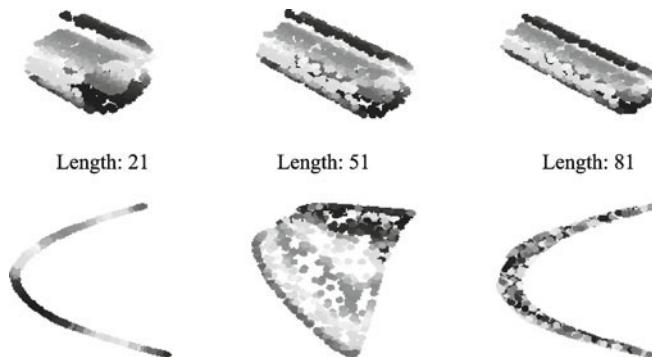


Fig. 12.5 The figure illustrates that when the graph neighborhood is inconsistent with manifold, the DR data may distort the geometric structure of dat so that it becomes invalid. Embeddings of the Swiss rolls are computed for different choices of the lengths of the roll, 21, 51, 81, respectively. The surface data are sampled in the same way as in Fig. 12.1. In the figure, the results of the experiment show that the dimensionality reduction for the Swiss roll becomes worse as the length increases. This phenomenon is due to the neighborhood inconsistency.

ric structure of the manifold. In Fig. 12.6, the different lengths of S-curve, 5, 10, 30, are chosen for the test with the same data settings as in Fig. 12.1. By the same argument we have made for the Swiss rolls in Fig. 12.5, the distortion of the DR result for the long S-curve is due to the neighborhood inconsistency.

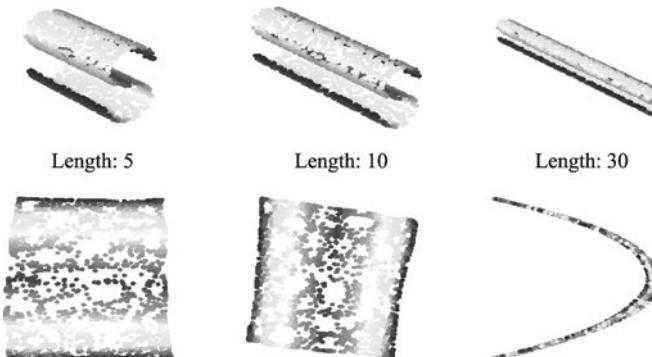


Fig. 12.6 The figure illustrates again that when the graph neighborhood is inconsistent with manifold, the DR data may distort the geometric structure of dat so that it becomes invalid. In this test, embeddings of the S-curves are computed for different choices of the lengths, 5, 10, and 30, respectively. The surface data are sampled in the same way as in Fig. 12.1. The results show that the dimensionality reduction becomes worse as the length increases, due to the neighborhood inconsistency.

12.3.2 Conclusion

Similar to LLE and LTSA, the Leigs kernel is sparse. The construction of the Leigs kernel is extremely simple if it adopts the adjacency matrix as the weight matrix. If the formula (12.15) or (12.25) is applied in the construction of the weight matrix, the computation is still straightforward and economic. Therefore, the algorithm performs very fast. However, Leigs method does not guarantee an isometric coordinate mapping h in DR processing. Hence, the reduced data may not preserve the local Euclidean distance. In some applications where local distance reservation is critical, the method may not provide the required DR. When a parametric weight matrix is used, the optimization of the parameter becomes a question although the algorithm is not very sensitive to it in general. Since the Leigs kernel is derived from a diffusion kernel, it relies on the smoothness of underlying manifolds. When the data resides on a non-smooth manifold, such as 3D-cluster, the method becomes invalid. Taking the advantage that the diffusion processing reduces noise, the Leigs algorithm tolerates a certain level of noise and may become unstable only if the noise is strong.

References

- [1] Belkin, M.: Problems of Learning on Manifolds. Ph.D. thesis, The University of Chicago (2003).
- [2] Belkin, M., Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation* 15(6), 1373–1396 (2003).
- [3] Belkin, M., Niyogi, P.: Semi-supervised learning on Riemannian manifolds. *Machine Learning* 56(1), 209–239 (2004).
- [4] Law, M.H.C., Jain, A.K.: Incremental nonlinear dimensionality reduction by manifold learning. *IEEE Trans. Pattern Analysis and Machine Intelligence* 28(3), 377–391 (2006).
- [5] Park, J.H., Zhang, Z.Y., Zha, H.Y., Kasturi, R.: Local smoothing for manifold learning. In: CVPR II:, pp. 452–459 (2004).
- [6] Yosida, K.: Functional Analysis. Springer-Verlag (1980).
- [7] Rosenberg, S.: The Laplacian on a Riemannian Manifold. Cambridge University Press (1997).
- [8] Donoho, D.L., Grimes, C.: Hessian eigenmaps: New locally linear embedding techniques for high-dimensional data. *Proc. Natl. Acad. Sci. USA* 100, 5591–5596 (2003).

Chapter 13 Hessian Locally Linear Embedding

Abstract Hessian locally linear embedding (HLLE) achieves linear embedding by minimizing the Hessian functional on the manifold where the data set resides. The conceptual framework of HLLE may be viewed as a modification of the Laplacian Eigenmaps framework. Let \mathcal{X} be the observed high-dimensional data which reside on a low-dimentional manifold M and h be the coordinate mapping on M so that $\mathcal{Y} = h(\mathcal{X})$ is a DR of \mathcal{X} . In Laplacian eigenmaps method, h is found in the numerically null space of the Laplace-Beltrami operator on M , while in Hessian locally linear embedding, it is found in the null space of the Hessian. Since HLLE embedding is locally linear, it works well for the data lying on a manifold that may not be convex. Compared with other nonlinear DR methods, such as Isomaps that need the data set lying on a convex manifold, HLLE can be applied to data in a wider range. The chapter is organized as follows. In Section 13.1, we describe the Hessian locally linear embedding method and its mathematical background. In Sections 13.2, the HLLE DR algorithm is introduced. The experiments of the algorithm are included in Section 13.3.

13.1 Description of Hessian Locally Linear Embedding

HLLE method was introduced by Donoho and Grimes [1]. While the (standard) LLE achieves linear embedding by minimizing the (squared) l_2 error in (10.2) in Section 10.1, the HLLE achieves linear embedding by minimizing the Hessian functional on the manifold where the data set resides. The conceptual framework of HLLE may be viewed as a modification of the Laplacian eigenmaps framework, which was introduced by Belkin and Niyogi [2, 3] (see Chapter 12). In HLLE, a quadratic form based on the Hessian is substituted in place of the original one based on the Laplacian. The following point perhaps gives the reason why Hessian is used to replace Laplacian: A linear function has everywhere vanishing Laplacian, but a function with everywhere vanishing Laplacian may not be linear, and a function is linear if and only if it has everywhere vanishing Hessian. By substituting the Hessian for the

Laplacian, we find a global embedding which is nearly linear in every set of local tangent coordinates. Since HLLE makes DR based on locally linearly embedding, it works well for the data lying on a manifold that may not be convex. Recall that some nonlinear DR methods, such as Isomaps, need the data set lying on a convex manifold.

We use the same data model as before. Let the data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ reside on a d -dimensional manifold $M \subset \mathbb{R}^D$. For each point $\mathbf{x} \in M$, there is a (manifold) neighborhood $O_{\mathbf{x}} \subset M$, which is differentially homeomorphic to an open set $U \subset \mathbb{R}^d$. Let $g = [g^1, \dots, g^D]' : U \rightarrow O_{\mathbf{x}}$ be a parameterization of $O_{\mathbf{x}}$. Then $h = [h^1, \dots, h^d]' = g^{-1} : O_{\mathbf{x}} \rightarrow U$ is a coordinate mapping on M . Of course, the parameterization g is not unique, and we may assume that g is an isometry from U to $O_{\mathbf{x}}$. Then $h = g^{-1}$ is an o.n. coordinate mapping on $O_{\mathbf{x}}$. We write $\mathbf{y} = h(\mathbf{x}) \in U$. Then $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_n\}$ with $\mathbf{y}_i = h(\mathbf{x}_i)$, $1 \leq i \leq n$, is a DR data set of \mathcal{X} .

13.1.1 Hessian on Manifold

The HLLE method achieves DR by setting \mathcal{Y} as the minimization of the Hessian functional on the manifold M . Let $f \in C^2(M) : M \rightarrow \mathbb{R}$ be a smooth function defined on M . To explain the idea more clearly, we first define the Hessian of f on M with respect to the manifold coordinates $h = [h^1, \dots, h^d]$.

Definition 13.1. Let $\{\mathbf{e}_i\}_{i=1}^d$ be the canonic coordinate basis of \mathbb{R}^d , $\mathbf{y} \in U \subset \mathbb{R}^d$, and $\mathbf{x} = g(\mathbf{y}) \in M$. For a function $f \in C^2(M)$, define

$$\frac{\partial f}{\partial y_i}(\mathbf{y}) = \lim_{t \rightarrow 0} \frac{f(\mathbf{y} + t\mathbf{e}_i) - f(\mathbf{y})}{t}.$$

Then the Hessian of f at \mathbf{x} with respect to the manifold coordinates $\mathbf{y} = h(\mathbf{x})$ is defined by the matrix

$$\mathbf{H}_{\mathbf{x}}^{\text{iso}}(f) = [\mathbf{H}_{\mathbf{x}}^{\text{iso}}(f)_{ij}]_{i,j=1}^d, \quad \mathbf{H}_{\mathbf{x}}^{\text{iso}}(f)_{ij} \stackrel{\text{def}}{=} \frac{\partial^2(f \circ g)}{\partial y_i \partial y_j}(\mathbf{y}),$$

and the corresponding Hessian functional on $C^2(M)$ is defined by

$$\mathcal{H}^{\text{iso}}(f) = \int_M \|\mathbf{H}_{\mathbf{x}}^{\text{iso}}(f)\|_{\text{F}}^2,$$

where $\|\cdot\|_{\text{F}}$ is the matrix Frobenius norm.

We shall call $H_{\mathbf{x}}^{\text{iso}}(f)$ and $\mathcal{H}^{\text{iso}}(f)$ manifold Hessian matrix and manifold Hessian functional respectively since they are evaluated on the manifold coordinates. For each coordinate function h^i , $h^i \circ g(\mathbf{y}) = y_i$. Hence, for any $\mathbf{x} \in M$, we have the following.

$$\mathbf{H}_{\mathbf{x}}^{\text{iso}}(h^i) = \mathbf{0}, \quad 1 \leq i \leq d, \tag{13.1}$$

$$\mathbf{H}_{\mathbf{x}}^{\text{iso}}(\mathbf{e}) = \mathbf{0}, \tag{13.2}$$

where e is the constant function on M such that $e(\mathbf{x}) = 1$. The pointwise relation of (13.1) and (13.2) can be extended to the global one.

$$\mathcal{H}^{\text{iso}}(f) = 0, \quad f = e, h^1, \dots, h^d. \quad (13.3)$$

As depicted in the discussion of Leigs method, the equations (13.3) hold only if the data set is in the idea model, i.e., \mathcal{X} strictly resides on M . In general, we shall modify the equations (13.3) to the following minimization problem.

$$h = \arg \min_{\langle \mathbf{F}, \mathbf{F} \rangle = \mathbf{I}} \mathcal{H}^{\text{iso}}(h), \quad \mathbf{F} = [e, h^1, \dots, h^d]', \quad h^i \in C^2(M), \quad (13.4)$$

where $\langle \mathbf{F}, \mathbf{F} \rangle = \mathbf{I}$ means that the function set $\{e, h^1, \dots, h^d\}$ forms an o.g. system in the space $C^2(M)$. The solution of (13.4) yields the DR data set \mathcal{Y} by $\mathbf{y}_i = h(\mathbf{x}_j)$.

13.1.2 Hessian on Tangent Space

Because the underlying manifold M is unknown, the parameterization g in the manifold Hessian functional $\mathcal{H}^{\text{iso}}(f)$ cannot be evaluated directly. To construct the HLLE DR kernel, we need to obtain a computable representation of $\mathcal{H}^{\text{iso}}(f)$. Let $T_{\mathbf{x}}M$ be the tangent space of M at \mathbf{x} and $L_{\mathbf{x}} = \mathbf{x} + T_{\mathbf{x}}M$ be the tangent hyperplane through the vector \mathbf{x} . We denote the o.g. projection from $O_{\mathbf{x}}$ to $L_{\mathbf{x}}$ by $P = P_{\mathbf{x}}$. Since $L_{\mathbf{x}}$ is close to $O_{\mathbf{x}}$, for each $\mathbf{p} \in O_{\mathbf{x}}$, $\mathbf{q} = P(\mathbf{p}) \in L_{\mathbf{x}}$ is an approximation of \mathbf{p} . We define an open set $N_{\mathbf{x}} \subset \mathbb{R}^D$ so that $O_{\mathbf{x}} \cup P(O_{\mathbf{x}}) \subset N_{\mathbf{x}}$. Let the function $f \in C^2(M)$ be smoothly extended on the open set $N_{\mathbf{x}} \subset \mathbb{R}^D$. For convenience, we shall denote the extension of f by itself.

We now choose an o.n. basis of $T_{\mathbf{x}}M$:

$$\mathcal{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_d\}, \quad \mathbf{b}_i \in \mathbb{R}^D.$$

Then each point $\mathbf{t} \in T_{\mathbf{x}}M$ has the tangent coordinate representation

$$\mathbf{t} = \sum_{i=1}^d t_i \mathbf{b}_i.$$

For the function $f \in C^2(M)$, which has been extended on $N_{\mathbf{x}} \subset \mathbb{R}^D$, we can compute its directional derivatives in the direction $\mathbf{b}_i \in T_{\mathbf{x}}M$:

$$\frac{\partial f}{\partial t_i}(\mathbf{x}) = \lim_{s \rightarrow 0} \frac{f(\mathbf{x} + s\mathbf{b}_i) - f(\mathbf{x})}{s}.$$

We now define the tangent Hessian functional of f at \mathbf{x} in the following.

Definition 13.2. The tangent Hessian matrix $\mathbf{H}_{\mathbf{x}}^{\tan}(f) = [\mathbf{H}_{\mathbf{x}}^{\tan}(f)_{ij}]$ of $f \in C^2(M)$ at \mathbf{x} (with respect to the tangent coordinates \mathbf{t}) is defined by

$$\mathbf{H}_{\mathbf{x}}^{\tan}(f)_{ij} = \frac{\partial^2 f}{\partial t_i \partial t_j}(\mathbf{x}), \quad (13.5)$$

and the corresponding Hessian functional in the tangent coordinates is defined by

$$\mathcal{H}^{\tan}(f) = \int_M \|\mathbf{H}_{\mathbf{x}}^{\tan}(f)\|_{\mathbf{F}}^2, \quad f \in C^2(M). \quad (13.6)$$

We shall simply call $\mathcal{H}^{\tan}(f)$ the tangent Hessian functional for convenience. The following theorem gives the equivalent relation of $\mathcal{H}^{\text{iso}}(f)$ and $\mathcal{H}^{\tan}(f)$.

Theorem 13.1. *The manifold Hessian functional of f is equal to the tangent Hessian functional of f :*

$$\mathcal{H}^{\tan}(f) = \mathcal{H}^{\text{iso}}(f).$$

Therefore, the minimization problem (13.4) can be reset as the following.

$$h = \arg \min_{\langle \mathbf{F}, \mathbf{F} \rangle = I} \mathcal{H}^{\tan}(h), \quad \mathbf{F} = [e, h^1, \dots, h^d]', \quad h^i \in C^2(M). \quad (13.7)$$

Since the tangent hyperplane $L_{\mathbf{x}}$ can be “learned” from the neighborhood of \mathbf{x} , the tangent Hessian functional is (approximately) computable. By Theorem 13.1, we may simply call either tangent Hessian functional or manifold Hessian functional *Hessian functional* and denote it by $\mathcal{H}(f)$. The proof of Theorem 13.1 is quite technical. We skip it here. Readers may refer to the appendix of [1].

13.1.3 Construction of Hessian Functional

We now study how to construct the discrete form of Hessian functional. A function f , restricted on \mathcal{X} , can be represented by the vector

$$\mathbf{f} = [f(\mathbf{x}_1), \dots, f(\mathbf{x}_n)]'.$$

Therefore the Hessian functional $\mathcal{H}(f)$ can be represented as the quadratic form of \mathbf{f} :

$$\mathcal{H}(f) = \mathbf{f}' \mathbf{K} \mathbf{f}, \quad (13.8)$$

where \mathbf{K} is an $n \times n$ positive semidefinite matrix, called HLLE kernel. The following theorem characterizes the Hessian functional.

Theorem 13.2. *The Hessian functional $\mathcal{H}(f)$ on $C^2(M)$ has a $d+1$ dimensional null space, which is spanned by the constant function and d manifold coordinate functions.*

Proof. The theorem can be proved by the direct computation. It is obvious that the constant function and all d coordinate functions are in the null space of the Hessian functional \mathcal{H} since all of them are linear. Assume that a function f is not linear, then at least one of its partial derivative of the second order does not vanish at a point, say, there is i, j , and a point \mathbf{y}_0 such that

$$\frac{\partial^2(f \circ g)}{\partial y_i \partial y_j}(\mathbf{y}_0) \neq 0.$$

Since the second partial derivative above is a continuous function, there is a neighborhood of \mathbf{y}_0 , say, $U_{\mathbf{y}_0}$, such that on $U_{\mathbf{y}_0}$

$$\frac{\partial^2(f \circ g)}{\partial y_i \partial y_j}(\mathbf{y}) \neq 0, \quad \mathbf{y} \in U_{\mathbf{y}_0}.$$

Therefore, $\mathcal{H}(f) > 0$. □

By Theorem 13.2, the function set $\{e, h^1, \dots, h^d\}$ is the set of 0-eigenfunctions of \mathbf{K} . In general cases, these functions are not exactly in the space, but certain deviations exist. Therefore, they are $d+1$ eigenvectors achieving the $d+1$ smallest eigenvalues of \mathbf{K} (see the model (13.7)).

To construct the HLLE kernel \mathbf{K} , we need the neighborhood structure on M . Assume that a neighborhood system is constructed on \mathcal{X} . We denote by $O_i = \{\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_k}\}$ the neighborhood of \mathbf{x}_i and by N_i the subscript set of O_i . Recall that all quadratic functions on d -dimensional vector space form a $d_q \stackrel{\text{def}}{=} (d+2)(d+1)/2$ dimensional function space. To ensure that all discrete quadratic functions on O_i are not degenerate, we set $k \geq d_q$. As usual, we assume also that the neighborhood system on \mathcal{X} is consistent with the neighborhood structure on M such that each O_i is a subset of the manifold neighborhood $W_i \subset M$ with $O_i = \mathcal{X} \cap W_i$ and $\cup_{i=1}^n W_i = M$. Let $\{\phi_i\}_{i=1}^m$ be a partition of unity of M with $\text{supp } \phi_i \subset W_i$. By the definition of integral on M , we have

$$\mathcal{H}(f) = \int_M \|\mathbf{H}_{\mathbf{x}}^{\tan}(f)\|_F^2 = \sum_{i=1}^m \int_{W_i} \phi_i \|\mathbf{H}_{\mathbf{x}}^{\tan}(f)\|_F^2. \quad (13.9)$$

Recall that the restriction of f on O_i has the local representation

$$\mathbf{f}_i = [f(\mathbf{x}_{i_1}), \dots, f(\mathbf{x}_{i_k})].$$

Therefore, the discrete form of the integral is the quadratic form

$$\mathcal{H}(f)|_{W_i} = \int_{W_i} \phi_i \|\mathbf{H}_{\mathbf{x}}^{\tan}(f)\|_F^2 = (\mathbf{f}_i)' \mathbf{W}_i \mathbf{f}_i, \quad (13.10)$$

where \mathbf{W}_i is a $k \times k$ positive semidefinite matrix whose null space consists of constant function and coordinate functions on the tangent space $T_{\mathbf{x}_i} M$.

To obtain tangent coordinates for $T_{\mathbf{x}_i} M$, we use the same method used in LTSA and Leigs. Let $\bar{\mathbf{x}} = \frac{1}{k} \sum_{j \in N(i)} \mathbf{x}_j$. Define the matrix

$$\mathbf{M}^i = [\mathbf{x}_{i_1} - \bar{\mathbf{x}}, \dots, \mathbf{x}_{i_k} - \bar{\mathbf{x}}], \quad (13.11)$$

and compute its SVD:

$$\mathbf{M}^i = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}' ,$$

where $\mathbf{U} \in \mathfrak{O}_{D,D}$, $\mathbf{V} \in \mathfrak{O}_{k,k}$ and $\boldsymbol{\Sigma}$ is a $D \times k$ diagonal matrix having the singular values on its diagonal, i.e.,

$$\boldsymbol{\Sigma} = \begin{cases} [\text{diag}(\sigma_1, \dots, \sigma_D), 0], & \text{if } k > D, \\ \text{diag}(\sigma_1, \dots, \sigma_D), & \text{if } k = D, \\ \begin{bmatrix} \text{diag}(\sigma_1, \dots, \sigma_k) \\ 0 \end{bmatrix}, & \text{if } k < D. \end{cases}$$

Let $\mathbf{U}^d = [\mathbf{u}_1, \dots, \mathbf{u}_d]$ be the first d columns extracted from \mathbf{U} and $\mathbf{V}^d = [\mathbf{v}_1, \dots, \mathbf{v}_d]$ be the first d columns extracted from \mathbf{V} . Since the dimension of $T_{\mathbf{x}_i} M$ is d , the set $\{\mathbf{u}_1, \dots, \mathbf{u}_d\}$ is an o.n. basis of $T_{\mathbf{x}_i} M$, and $\mathbf{v}_1, \dots, \mathbf{v}_d$, are the discrete tangent coordinate functions on O_i . By (13.11), the local data matrix \mathbf{M}^i is centered so that $\langle \mathbf{v}_j, \mathbf{1} \rangle = 0$, $1 \leq j \leq d$. Write

$$\mathbf{L}^i = [\mathbf{1}, \mathbf{V}^d].$$

The columns of \mathbf{L}^i form an o.n. basis for the space of linear functions on O_i .

Let a and b be two functions on $T_{\mathbf{x}_i} M$. Their restrictions on O_i are vectors $\mathbf{a} = [a_1, \dots, a_k]'$ and $\mathbf{b} = [b_1, \dots, b_k]'$, where $a_j = a(\mathbf{x}_{i_j})$ and $b_j = b(\mathbf{x}_{i_j})$. We denote the discrete form of the product function ab by the Hadamard product $\mathbf{a} \boxtimes \mathbf{b}$:

$$\mathbf{a} \boxtimes \mathbf{b} = [a_1 b_1, \dots, a_k b_k]'$$

Then the column vectors in the matrix

$$\mathbf{Q}^i = [\mathbf{v}_l \boxtimes \mathbf{v}_j]_{1 \leq l \leq j \leq d}$$

form a basis of the space of all homogeneous quadratic functions on O_i and the columns of the $d_q \times k$ matrix

$$\mathbf{B}^i = [\mathbf{L}^i \ \mathbf{Q}^i]$$

form a basis of the space of all quadratic functions on O_i . We orthonormalize \mathbf{B}^i to

$$\mathbf{B}_{\text{on}}^i = [\mathbf{L}^i \ \mathbf{Q}_{\text{on}}^i]$$

so that all columns of \mathbf{B}_{on}^i form an o.n. basis of the space of quadratic functions. Particularly, the columns of the matrix

$$\mathbf{Q}_{\text{on}}^i = [\mathbf{q}_1 \ \dots \ \mathbf{q}_r], \quad r = d(d+1)/2,$$

form an o.n. basis of the space of all homogeneous quadratic functions. We define

$$\mathbf{H}^i = (\mathbf{Q}_{\text{on}}^i)'.$$

Then

$$\begin{aligned}\mathbf{H}^i \mathbf{e} &= \mathbf{0}, \\ \mathbf{H}^i \mathbf{v}_j &= \mathbf{0}, \quad j = 1, \dots, d,\end{aligned}$$

and

$$\mathbf{H}^i \mathbf{q}_j = \mathbf{e}_j,$$

where \mathbf{e}_j is the j th unit vector. We claim that the discrete local Hessian \mathbf{W}_i in (13.10) has the representation

$$\mathbf{W}_i = (\mathbf{H}^i)' \mathbf{H}^i, \tag{13.12}$$

for we have

$$\begin{aligned}\mathbf{1}' \mathbf{W}_i \mathbf{1} &= 0, \\ \mathbf{v}_j' \mathbf{W}_i \mathbf{v}_j &= 0, \quad 1 \leq j \leq d, \\ \mathbf{q}_j' \mathbf{W}_i \mathbf{q}_l &= \delta_{lj}, \quad 1 \leq j \leq l \leq d.\end{aligned}$$

13.1.4 Construct of HLLE DR Kernel

To construct the HLLE DR kernel \mathbf{K} in (13.8), we apply the formula (13.9). Adopting the same trick used in LTSA, we extend the local formula (13.10) to the global one as follows. Let \mathbf{W}^i be the $n \times n$ matrix such that all of its entries are zero except that

$$\mathbf{W}^i(N(i), N(i)) = \mathbf{W}_i,$$

where $\mathbf{W}^i(N(i), N(i))$ denote the $k \times k$ submatrix of \mathbf{W}^i with both row indices and column indices taken from $N(i)$. Then the matrix

$$\mathbf{K} = \sum_{i=1}^n \mathbf{W}^i \tag{13.13}$$

is the HLLE kernel.

13.2 HLLE Algorithm

13.2.1 HLLE Algorithm Description

In HLLE algorithm, the given data $\mathcal{X} \subset \mathbb{R}^D$ is the input. Assume that the dimension of the DR data is d and the DR set is denoted by $\mathcal{Y} \subset \mathbb{R}^d$. Then the output of the algorithm is the data matrix \mathbf{Y} . The HLLE algorithm consists of the following steps.

Step 1. Neighborhood definition. Either k -neighborhood or ε -neighborhood can be used for defining the neighborhood system on the observed data set \mathcal{X} . In most cases, k -neighborhood is used. Let $r = (d+2)(d+1)/2$. The HLLE algorithm employs the neighborhood size $k \geq r$ in the computation of local Hessian functionals.

Step 2. Local tangent coordinate functional creation. First the coordinates of the tangent space at each point \mathbf{x}_i are estimated by PCA on its neighborhood. Let the local data set be $\mathcal{X}^i = \{\mathbf{p}_{j_1}, \dots, \mathbf{p}_{j_k}\}$. Apply the PCA algorithm to obtain d principal components of \mathcal{X}^i in the $k \times d$ matrix $\mathbf{V}^i = [\mathbf{v}_1 \dots \mathbf{v}_d]$. Then the columns of \mathbf{V}^i are tangent coordinate functions on \mathcal{X}^i .

Step 3. Local Hessian functional construction. Let $\mathbf{1} = [1, \dots, 1]' \in \mathbb{R}^k$ and $\mathbf{Q}^i = [\mathbf{v}_i \boxtimes \mathbf{v}_j]_{1 \leq i \leq j \leq d}$. Define

$$\mathbf{V}^a = [\mathbf{1}, \mathbf{V}^i, \mathbf{Q}^i].$$

Apply the Gram-Schmidt procedure on \mathbf{V}^a to obtain its orthonormalization $[\mathbf{1}, \mathbf{V}^i, \tilde{\mathbf{Q}}^i]$. Then the local Hessian functional is $\mathbf{W}_i = \tilde{\mathbf{Q}}^i(\tilde{\mathbf{Q}}^i)'$.

Step 4. HLLE DR kernel construction. Initialize the kernel \mathbf{K} to an $n \times n$ zero matrix. Update \mathbf{K} by $\mathbf{K}(N(i), N(i)) = \mathbf{K}(N(i), N(i)) + \mathbf{W}_i$, where $\mathbf{K}(N(i), N(i))$ denote the submatrix of \mathbf{K} containing both rows and columns with the indices in $N(i)$.

Step 5. Eigen decomposition of HLLE DR kernel. Let $\{Y^0, Y^1, \dots, Y^d\} \subset \mathbb{R}^n$ be the $d+1$ eigenvectors corresponding to the $d+1$ smallest ascending eigenvalues of \mathbf{K} . Then the DR date set is $\mathbf{Y} = [Y^1; \dots; Y^d]'$.

Remark 13.1. In the original HLLE algorithm (see basis.stanford.edu/HLLE) developed by the authors of [1], a weight matrix \mathbf{W} of the size $d_p N \times N$ is construct, then the kernel is obtained by the matrix product, $\mathbf{K} = \mathbf{W}\mathbf{W}'$. We simplify the kernel construction in Steps 3 and 4 using $\mathbf{W} = \sum_{i=1}^n \mathbf{W}_i$.

13.2.2 Matlab Code of HLLE

The Matlab code of HLLE is given in the following.

```

function [Y, mse] = drhlle(X, d, options)
% Hessian Locally Linear Embedding for DR.
% INPUTS:
%   X: D x N data matrix, X(:,i) is ith point.
%   d: Dimension of output data.
%   OPTIONS:
%     .epsilon: epsilon-neighbors.
%     .k: k-neighbors. k can be either a vector
%         or a scalar number.
%     .verb: display the processing verbally.
% OUTPUT:
%   Y: d x N dimension-reduced data matrix.
%   The large eigenvector is on the top.
%   MSE: N-vector.
%       The sum of the eigenvalues(d+2:end)
% (at each neighborhood used) of the local
%       coordinate representation, used for
%       adaptive neighborhood restructuring.
% Example:
% N=1000; options.k=12; d=2;
% tt = (3*pi/2)*(1+2*rand(1,N));
% height = 21*rand(1,N);
% X = [tt.*cos(tt); height; tt.*sin(tt)];
% [Y, eigv, mse] = DRHLLE(X,d,options);
%
% CALL: data_neighborhood
%
% Code is based on HLLE method.
% Reference:
% D.L. Donoho and C. Grimes, (2003)
% "Hessian eigenmaps: Locally linear embedding
% techniques for high-dimensional data."
% Proc. Nat. Acad. Sci., vol. 100, pp.5591-5596.
%
% The code is written by Jianzhong Wang, 2008.
%
% Initialization.
N = size(X,2);
% initialize Kernel.
G = sparse(N,N);
% initialize Energy loss vector.
mse = zeros(N,1);
% Initialize options.
options.null=0;
if ~isfield(options, 'epsilon')
    if isfield(options, 'k')

```

```

k = options.k;
if numel(k)==1
    k = k*ones(N,1);
end
else
    k = 10*ones(N,1);
end
end
if isfield(options, 'verb')
    verb = option.berb;
else
    verb = 1;
end
% Step 1: Construct data-graph.
if verb
    fprintf(1,'--Construct data graph\n');
end
[D2, nghd] = data_neighborhood(X,options);
% find # of points for epsilon-neighbors.
if isfield(options,'epsilon')
    k = sum(nghd,2);
end
% Step 2: Construct HLLE kernel.
if verb
    fprintf(1,'--Construct HLLE kernel\n');
end
% regularization in case constrained fits
% are ill conditioned.
if(mean(k)>d)
    if verb
        disp('If mean(k)>d, regularization is used.');
    end
    options.tol=1e-3;
else
    options.tol=0;
end
% Kernel construcion.
for i=1:N
    %take k nearest neighbors.
    thisi = find(nghd(:,i));
    thisx = X(:, thisi);
    % centralize data
    thisx = thisx - repmat(mean(thisx,2),1,k(i));
    % compute local tangent coordinate functions.
    [U,D,Vpr] = svd(thisx);
    V = Vpr(:,1:d);
    vals = diag(D);
    mse(i) = sum(vals(d+1:end));
    % build quadratic functions.
    Yi = ones(k(i),(d+1)*(d+2)/2);

```

```

Yi(:,2:d+1)= V;
ct = d+1;
for mm = 1:d
    nn=1:(d+1-mm);
    Yi(:,ct+nn) = V(:,nn).*V(:,mm:d);
    ct = ct+d-mm+1;
end
% make o.n. basis for quadratic function space.
Yt = mgs(Yi);
Pi = Yt(:,d+2:end);
% Make the HLLE kernel by extending local one.
G(thisi,thisi) = G(thisi,thisi)+Pi*Pi';
end
clear X
%Step 3. Eigen decomposition
if verb
    fprintf(1,'--Find dimension-reduced data.\n');
end
options.disp = 0;
options.isreal = 1;
options.issym = 1;
[Y, eigv] = eigs(G,d+1,'SM',options);
% bottom evect is [1,1,1,...] with eval 0.
% Scale the output
Y = Y(:,1:d)/*sqrt(N);
R = Y'*Y;
R2 = R^(-1/2);
Y = Y*R2;
%
% Auxiliary Functions
%
function [Q, R] = mgs(A)
% Modified Gram-Schmidt, a more stable way
% to compute a QR factorization.
n = size(A,2);
% Assume that m>= n.
R = zeros(n,n);
for i= 1:n-1
    R(i,i) = norm(A(:,i));
    A(:,i) = A(:,i)/R(i,i);
    for j = i+1:n
        R(i,j) = A(:,i)' * A(:,j);
        A(:,j) = A(:,j) - R(i,j) * A(:,i);
    end
end
R(n,n) = norm(A(:,n));
A(:,n) = A(:,n)/R(n,n);
Q = A;

```

13.3 Implementation of HLLE

13.3.1 Experiments on Artificial Surfaces

In this section, we demonstrate the HLLE method in the dimensionality reduction for artificial surfaces. For comparing HLLE method with other nonlinear DR methods, we adopt the same data settings as in LLE, LTSA, and Leigs in the experiments.

In Fig. 13.1, 3-dimensional data of 4 artificial surfaces, 3D-cluster, Punched sphere, Swiss roll, and S-curve are reduced to 2-dimensional ones by HLLE algorithm. In the experiment, 1000 points are randomly sampled on each surface and 10 nearest points are used to construct the neighborhood of a point. The experiment shows that HLLE algorithm works well for S-curve, Swiss roll, and Punched sphere. It cannot produce a valid DR for 3D-cluster. Note that HLLE is based on the minimization of Hessian functional, which is defined on a smooth manifold. Since 3D-cluster is not a smooth surface, HLLE performs poorly on it. HLLE adopts a local isometric manifold coordinate mapping, which well preserves the local Euclidean distance for the data. Hence, HLLE has a good DR result for Swiss roll too. Noise impacts the dimensionality reduction results. In Fig. 13.2 and Fig. 13.3, we apply HLLE algorithm to reduce the dimension of noisy data, with the noise standard

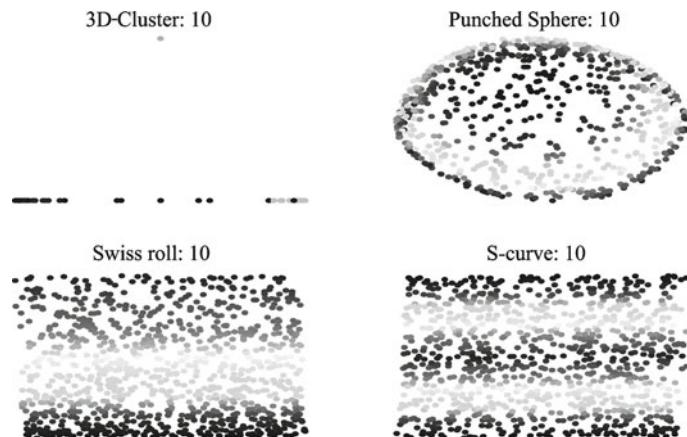


Fig. 13.1 HLLE algorithm is tested on 4 surfaces: 3D-cluster, Punched sphere, Swiss roll, and S-curve. 1000 points are randomly sampled on each of them. All are reduced to 2-dimensional data (on the plane). The original 3-dimensional graphs of these surfaces are not presented here. They can be found in Section 1.5. The figure displays the DR results for these surfaces. The graph shows that HLLE algorithm works well for them except 3D-cluster.

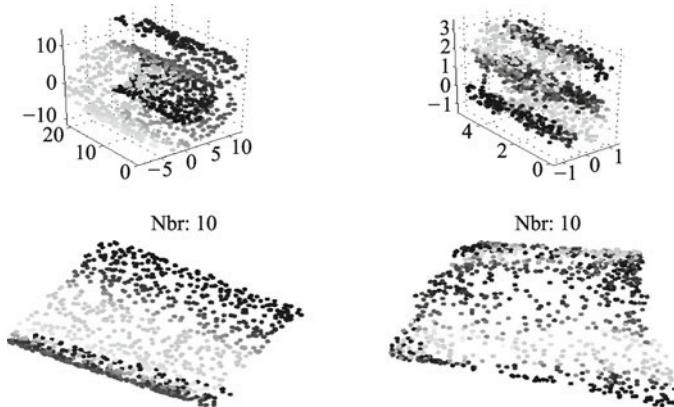


Fig. 13.2 HLLE algorithm is applied on two surfaces, Swiss roll and S-curve, with the noise standard deviation 0.2. All of the surfaces are sampled in the same way as in Fig. 13.1. We do not show the DR results for 3D-cluster and Punched sphere since they are not valid at this noise level. The figure shows that HLLE algorithm is very sensitive to noise. At the top line, two noisy data are presented. Top left: Swiss roll. Top right: S-curve. At the bottom line are their corresponding dimensional reduced 2-dimensional data.

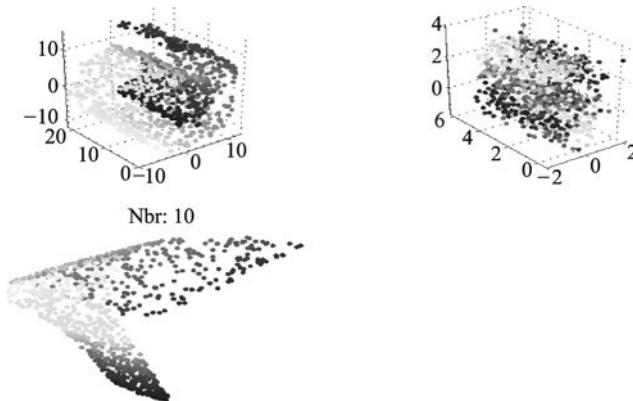


Fig. 13.3 HLLE algorithm is applied on two surfaces, Swiss roll and S-curve, with the noise standard deviation 0.4. The surfaces are sampled in the same way as in Fig. 13.1. The figure shows that when the noise deviation is large, the DR results of HLLE algorithm have large distortions. At the top line, two noisy data are presented. Top left: Swiss roll. Top right: S-curve. At the bottom line are their corresponding dimensional reduced 2-dimensional data. There no output for S-curve because HLLE becomes very unstable so that it cannot produce a valid result

deviation 0.2 and 0.4 respectively. The results show that HLLE algorithm is sensitive to noise, performing worse than Leigs and Isomaps. This is perhaps because the Hessian functional in HLLE requires smoothness of the surface. Recall that the HLLE DR kernel is derived from tangent Hessian functional.

When noise exists, the DR data may not preserve the data geometry very well since the data having a large deviation distort the smoothness. As we discussed before, the properties of the underlying manifold greatly impact the DR results. In Fig. 13.4, the different heights of Punched sphere are chosen for the test. It is not surprise that when the height increases, the DR results have more distortions. Compared with LTSA, LLE, and Leigs, HLLE has a worse performance since it wants to preserve the local distances. In Fig. 13.5 HLLE algorithm is applied to Swiss rolls with lengths 21, 51, and 81, respectively. The results show that when the length is 81, the DR result

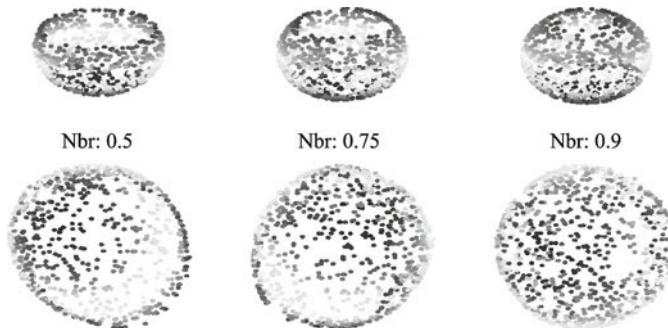


Fig. 13.4 Effect of surface shape on HLLE. Embeddings of the punched spheres are computed for the heights of the punched sphere, 0.5, 0.75, and 0.9, respectively, where the diameter of the original sphere is unit. It shows that the dimensionality reduction for the punched sphere becomes worse as the height increases.

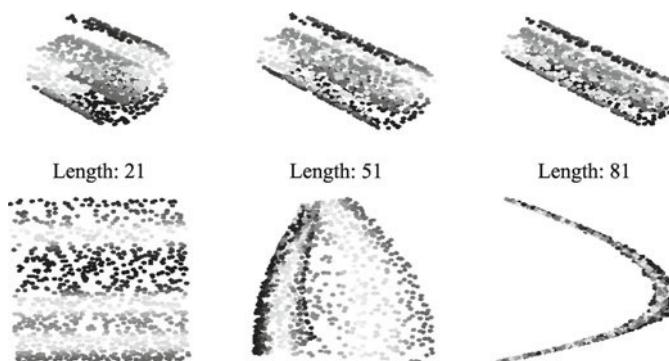


Fig. 13.5 The figure illustrates that when the graph neighborhood is inconsistent with manifold, the DR data may distort the geometric structure of dat so that it becomes invalid. Embeddings of the Swiss rolls are computed for different choices of the lengths of the roll, 21, 51, 81, respectively. The surface data are sampled in the same way as in Fig. 13.1. In the figure, the results of the experiment show that the dimensionality reduction for the Swiss roll becomes worse as the length increases. This phenomenon is due to the neighborhood inconsistence, not method deficiency.

is not valid. This phenomenon can be explained by the same reason we have given to Leigs algorithm in Chapter 12. In Fig. 13.6, the lengths of S-curve, 5, 10, 30, are chosen for the test with the same data settings as in Fig. 13.1. By the same argument we have made for the Swiss rolls in Fig. 13.5, the distortion of the DR result for the long S-curve is due to the neighborhood inconsistence.

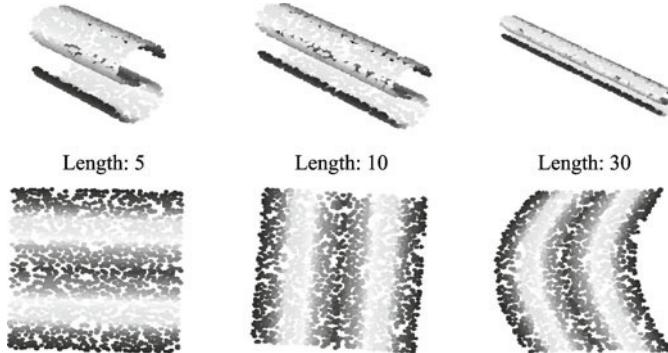


Fig. 13.6 The figure illustrates again that when the graph neighborhood is inconsistent with manifold, the DR data may distort the geometric structure of dat so that it becomes invalid. In this test, embeddings of the S-curves are computed for different choices of the lengths, 5, 10, and 30, respectively. The surface data are sampled in the same way as in Fig. 13.1. The results show that the dimensionality reduction becomes worse as the length increases, due to the neighborhood inconsistence.

HLLE algorithm is relatively insensitive to the neighborhood sizes. It is stable over a wide range of neighborhood sizes. The size of that range depends on various features of the data, such as the sampling density and the manifold geometry. Our experiment shows that when we perform HLLE for the mentioned 4 surfaces, it is stable if the neighborhood size is less than 7. It is also shown that when the size is large than 12, the DR data distort the local geometry of the original dat. The detailed results are included in Fig. 13.7. The last experiment illustrates that HLLE can work well for the data on a non-convex manifold. Here a change in sampling procedure for Swiss roll is considered. Instead of sampling parameters in a full rectangle, we sample from a rectangle with a missing rectangular strip punched out of the center. The resulting Swiss roll is then missing the corresponding strip and thus is not convex (while still remaining connected). Using this data set we test Isomap [4, 5], LLE [6, 7], and HLLE [1], respectively, on a random sample of 600 points in three dimensions. The results, as seen in Fig. 13.8 (copied from [1]), show the dramatic effect that nonconvexity can have on the resulting embeddings. Although the data manifold is still locally isometric to Euclidean space, the effect of the missing sampling region is, in the case of LLE, to make the resulting embedding functions asymmetric and nonlinear

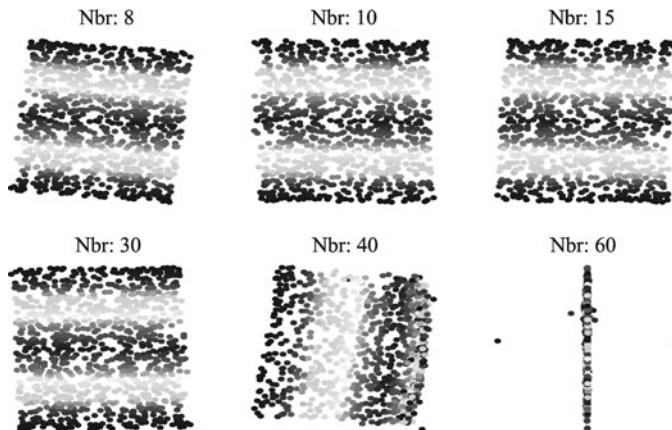


Fig. 13.7 Effect of neighborhood size on HLLE. HLLE is relatively insensitive to the neighborhood size. In this figure, HLLE is applied to Swiss roll for different choices of the number of nearest neighbors k , 8, 10, 12, 15, 30, 60, respectively. The algorithm produces a distorted DR result only when the neighborhood size is 60. It is similar to LLE, A reliable HLLE embedding from 3-dimension to 2-dimension can be obtained over a wide range of values.

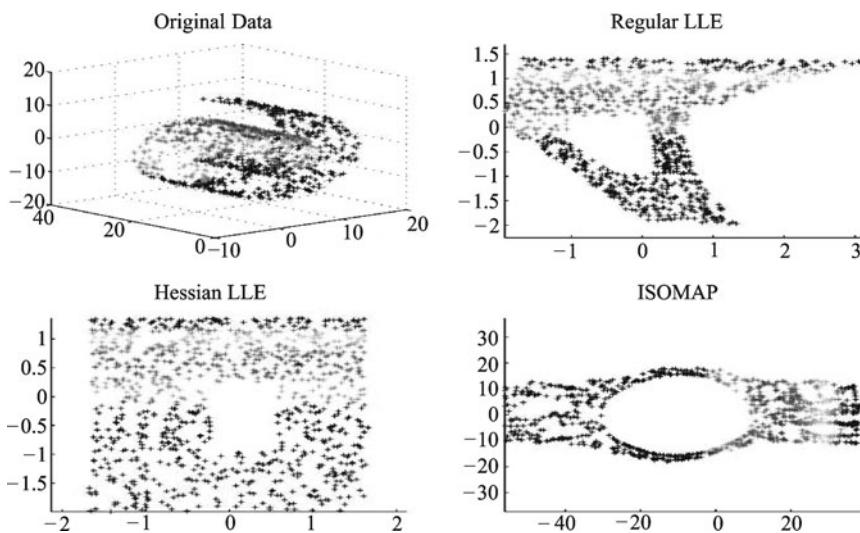


Fig. 13.8 HLLE is valid for the nonconvex data. Top left: Original data. Top right: LLE embedding with $k = 12$. Bottom left: HLLE with $k = 12$. Bottom right: Isomap with $k = 7$. The underlying correct parameter space that generated the data is a square with a central square removed, similar to what is obtained by the Hessian approach (Bottom left). The graph is copied from the original in [1].

with respect to the original parameterization. In the case of Isomap, the nonconvexity causes a strong dilation of the missing region, warping the rest of the embedding. Hessian LLE, on the other hand, embeds the result almost perfectly into two-dimensional space.

13.3.2 Conclusion

The HLLE kernel is sparse, and the HLLE algorithm works well even when the underlying manifold M is not convex. In addition, the HLLE algorithm is a fast algorithm that allows high accuracy. However, the Hessian functional involves the second derivatives, so that the algorithm is quite sensitive to noise.

References

- [1] Donoho, D.L., Grimes, C.: Hessian eigenmaps: New locally linear embedding techniques for high-dimensional data. *Proc. Natl. Acad. Sci. USA* 100, 5591–5596 (2003).
- [2] Belkin, M.: Problems of Learning on Manifolds. Ph.D. thesis, The University of Chicago (2003).
- [3] Belkin, M., Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation* 15(6), 1373–1396 (2003).
- [4] de Silva, V., Tenenbaum, J.B.: Global versus local methods in nonlinear dimensionality reduction. In: S. Becker, S. Thrun, K. Obermayer (eds.) *Neural Information Processing Systems (NIPS 2002)*, pp. 705–712. MIT Press (2002).
- [5] Tenenbaum, J.B., de Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. *Science* 290(5500), 2319–2323 (2000).
- [6] Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. *Science* 290(5500), 2323–2326 (2000).
- [7] Saul, L.K., Roweis, S.T.: Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research* 4, 119–155 (2003).

Chapter 14 Diffusion Maps

Abstract In Laplacian eigenmaps method, the DR data is obtained from the eigen-subspace of the Laplace-Beltrami operator on the underlying manifold where the observed data resides. In Chapter 12, it was pointed out that Laplace-Beltrami operator directly links up with the heat diffusion operator by the exponential formula for positive self-adjoint operators. Therefore, they have the same eigenvector set, and the corresponding eigenvalues are linked by the exponential relation too. The relation indicates that the diffusion kernel on the manifold itself can be used in the construction of DR kernel. The diffusion map method (Dmaps) constructs the DR kernel using the diffusion maps. Then the DR data is computed from several leading eigenvectors of the Dmaps DR kernel. The chapter is organized as follows. In Section 14.1, we describe Dmaps method and its mathematical background. In Section 14.2, we present Dmaps algorithms with different types of normalization. The implementation of the various Dmaps algorithms is included in Section 14.3. In Section 14.4, we discuss the applications of Dmaps in the extraction of data features. In Section 14.5, several results of the implementation of Dmaps feature extractors are displayed.

14.1 Description of DR Method of Diffusion Maps

The Gaussian-type diffusion kernels were widely used in machine learning and data clusters before 2004 [1–7]. The mathematics of diffusion maps was first studied by Coifman and Lafon [8, 9]. The further discussion and applications can be found in [10–12]. The idea of the Dmaps method is to consider a family of diffusion maps, each of which embeds the data set into a Euclidean space so that the Euclidean distance in the space is equal to the diffusion distance on the data. The various diffusion distances induced by diffusion maps describe the multiscale geometric structures on the data, represented by feature functions. Among them, one truly reveals the targeted features and the corresponding embedding produces the required dimensional reduction of the original data. In this chapter, we adopt the same data model

as before, assuming that the data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ resides on a d -dimensional manifold $M \subset \mathbb{R}^D$, which has a parameterization g . Let $h \approx g^{-1} = [h^1, \dots, h^d]'$ be the feature mapping on M . The DR data set is obtained as the feature representation of \mathcal{X} : $\mathcal{Y} = h(\mathcal{X})$.

14.1.1 Diffusion Operator on Manifold

The DR method of diffusion maps has the similar idea as the method of Laplacian eigenmaps. Hence, we first review the relation between Laplace-Beltrami operator \mathcal{L} and the Neumann heat diffusion operator \mathcal{A}_t . Recall that the Laplace-Beltrami operator \mathcal{L} and the Neumann heat operator \mathcal{A}_t on M are linked by the exponential formula

$$\mathcal{A}_t = e^{-t\mathcal{L}}. \quad (14.1)$$

Since \mathcal{A}_t has the integral representation

$$\mathcal{A}_t(f)(\mathbf{x}) = \int_M G_t(\mathbf{x}, \mathbf{y}) f(\mathbf{y}), \quad f \in C(M), \quad (14.2)$$

it can be extended onto a larger function space, say, it can be considered as an operator on $L^2(M)$. Let the inner product on $L^2(M)$ be denoted by $\langle \cdot, \cdot \rangle$. The spectral decomposition of \mathcal{L} leads to

$$\mathcal{L}(f) = \sum_{i=0}^{\infty} \lambda_i \phi_i^i \langle \phi_i^i, f \rangle, \quad (14.3)$$

where

$$0 = \lambda_0 \leq \lambda_1 \leq \dots \leq \lambda_n \rightarrow \infty$$

are eigenvalues of \mathcal{L} and $\phi_i, 0 \leq i < \infty$, are corresponding eigenfunctions. Unfortunately, the operator \mathcal{L} is unbounded on $L^2(M)$. By (14.3) and (14.1), the spectral decomposition of \mathcal{A}_t is

$$\mathcal{A}_t(f) = \sum_{i=0}^{\infty} e^{-t\lambda_i} \phi_i^i \langle \phi_i^i, f \rangle, \quad (14.4)$$

which shows that the Neumann heat diffusion operator \mathcal{A}_t has the same eigenfunction set as the Laplace-Beltrami operator \mathcal{L} , but \mathcal{A}_t is a bounded operator on $L^2(M)$ with the operator norm 1. Besides, all of its eigenvalues are nonnegative so that it is also a positive self-adjoint operator. These important properties make diffusion operators widely used in many applications. Dmaps method uses the eigen decomposition of \mathcal{A}_t to find the DR data set. In practice, we use the following maximization model. For a certain t ,

$$h = \arg \max \langle f, \mathcal{A}_t(f) \rangle, \quad \text{s.t. } \mathbf{F}' \mathbf{F} = \mathbf{I}, \quad (14.5)$$

where the same as in Leigs, we set $\mathbf{F} = [e, h^1, \dots, h^d]$.

14.1.2 Normalization of Diffusion Kernels

The construction of Dmaps kernel is very similar to the construction of Leigs kernel. Dmaps kernel is derived from the Neumann heat kernel, which can be approximated by Gaussian. Hence, a Dmaps kernel can be constructed by normalizing the Gaussian. In Chapter 12, we already learned that the normalization of Gaussian in the continuous data model is independent of variables \mathbf{x} , but depends on \mathbf{x} in the discrete data model. In the normalization of discrete Gaussian we are caught in a dilemma: if we want to keep the sum of each row in the kernel being 1 so that $\mathbf{1}$ is the 1-eigenvector of the normalized kernel, then we lose the self-adjoint property of the kernel.

In diffusion maps, we construct symmetric Dmaps kernel, replacing the row-sum-one rule by a weaker rule: 1 is the greatest eigenvalue of the kernel. We construct two different types of Dmaps kernels: Graph-Laplacian type and Laplace-Beltrami type. Let A_t be a non-normalized Gaussian, say,

$$A_t(\mathbf{x}, \mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{t}},$$

which generates a bi-linear functional $\langle g, \mathcal{A}(f) \rangle$ on $L^2(M) \times L^2(M)$ having the form

$$\langle g, \mathcal{A}(f) \rangle = \int_M \int_M A_t(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) g(\mathbf{y}).$$

Write

$$S(\mathbf{x}) = \int_M A_t(\mathbf{x}, \mathbf{y}) d\mathbf{y}$$

and normalize $A_t(\mathbf{x}, \mathbf{y})$ by

$$\tilde{A}_t(\mathbf{x}, \mathbf{y}) = \frac{A_t(\mathbf{x}, \mathbf{y})}{S(\mathbf{x})}.$$

Then $\tilde{A}_t(\mathbf{x}, \mathbf{y})$ has row-sum-one property:

$$\int_M \tilde{A}_t(\mathbf{x}, \mathbf{y}) d\mathbf{y} = 1.$$

But $\tilde{A}_t(\mathbf{x}, \mathbf{y})$ is not symmetric. The kernel \tilde{A}_t generates the operator

$$\tilde{\mathcal{A}}_t(f) = \int_M \tilde{A}_t(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{y}.$$

To construct a symmetric kernel, we set

$$\hat{A}_t(\mathbf{x}, \mathbf{y}) = \frac{A_t(\mathbf{x}, \mathbf{y})}{\sqrt{S(\mathbf{x})S(\mathbf{y})}},$$

which generates the operator

$$\hat{\mathcal{A}}_t(f) = \int_M \hat{A}_t(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{y}.$$

We can see that the operators $\tilde{\mathcal{A}}_t$ and $\hat{\mathcal{A}}_t$ have the same eigenvalue set. Furthermore, f is an eigenvector of $\tilde{\mathcal{A}}_t$ achieving the eigenvalue λ if and only if $\frac{f}{\sqrt{S}}$ is the eigenvector of $\hat{\mathcal{A}}_t$ achieving the same eigenvalue. This conclusion has a simple proof as follows. If we have

$$\tilde{\mathcal{A}}_t(f) = \lambda f,$$

then

$$\begin{aligned}\hat{\mathcal{A}}_t\left(\frac{f}{\sqrt{S}}\right)(\mathbf{x}) &= \frac{1}{\sqrt{S(\mathbf{x})}} \int_M A_t(\mathbf{x}, \mathbf{y}) \frac{f(\mathbf{y})}{S(\mathbf{y})} d\mathbf{y} \\ &= \frac{1}{\sqrt{S(\mathbf{x})}} \tilde{\mathcal{A}}_t(f)(\mathbf{x}) \\ &= \lambda \left(\frac{f}{\sqrt{S}}\right)(\mathbf{x}),\end{aligned}$$

which indicates that $\frac{f}{\sqrt{S}}$ is the eigenfunction of $\hat{\mathcal{A}}_t$ achieving λ . The proof of the reverse is similar. Since this normalization is similar to that for graph Laplacian, we call \hat{A}_t the diffusion kernel of *Graph-Laplacian type*. Note that \sqrt{S} is the eigenfunction of $\hat{\mathcal{A}}_t$ achieving 1.

Another type of normalization of A_t employs the symmetric kernel

$$A_t^{\text{sym}}(\mathbf{x}, \mathbf{y}) = \frac{A_t(\mathbf{x}, \mathbf{y})}{S(\mathbf{x})S(\mathbf{y})}.$$

Let

$$S^{\text{sym}}(\mathbf{x}) = \int_M A_t^{\text{sym}}(\mathbf{x}, \mathbf{y}) d\mathbf{y},$$

which normalizes A_t^{sym} to the Dmaps kernel

$$\bar{A}_t = \frac{A_t^{\text{sym}}(\mathbf{x}, \mathbf{y})}{\sqrt{S^{\text{sym}}(\mathbf{x})S^{\text{sym}}(\mathbf{y})}}, \quad (14.6)$$

corresponding to the operator

$$\bar{\mathcal{A}}_t(f) = \int_M \bar{A}_t(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{y}.$$

Because the infinitesimal of $\bar{\mathcal{A}}_t$ is \mathcal{L} :

$$s - \lim_{t \rightarrow 0} \frac{\bar{\mathcal{A}}_t - I}{t} = \mathcal{L},$$

we call \bar{A}_t the diffusion kernel of *Laplace-Beltrami type*. The function $\sqrt{S^{\text{sym}}}$ is the eigenfunction of $\bar{\mathcal{A}}_t$ achieving 1. The discretization of the kernels \hat{A}_t

and \bar{A}_t can be trivially derived too. We skip it here.

14.2 Diffusion Maps Algorithms

14.2.1 Dmaps DR Algorithm Description

Let given data $\mathcal{X} \subset \mathbb{R}^D$ be the input of a Dmaps algorithm. Assume that the dimension of the DR data is d . Then the output of the algorithm is the DR data $\mathcal{Y} \subset \mathbb{R}^d$ in the matrix form. A Dmaps algorithm consists of the following steps.

Step 1. Data graph construction. This step is same as other nonlinear DR methods. Either k -neighborhood or ε -neighborhood can be used for defining data graph.

Step 2. Weight matrix creation. Assume that a date graph $[\mathcal{X}, \mathbf{A}]$ is defined on the data set \mathcal{X} , which determines a neighborhood system on \mathcal{X} . Let O_i denote the neighborhood of \mathbf{x}_i and $N(i)$ denote the corresponding subscript set, which is read from the i th row of \mathbf{A} . We create the weight matrix $\mathbf{W} = [w_{ij}]_{i,j=1}^n$ by setting

$$w_{ij} = \begin{cases} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{t}\right), & j \in N(i) \cup \{i\}, \\ 0, & \text{otherwise,} \end{cases} \quad (14.7)$$

where $t > 0$ is a parameter. To create a non-parameter weight matrix, we choose

$$t = \frac{1}{\sum_{i=1}^n |N(i)|} \sum_{i=1}^n \sum_{j \in N(i)} \|\mathbf{x}_i - \mathbf{x}_j\|^2.$$

Step 3. Diffusion kernel construction. We normalize the weight matrix to construct the Dmaps kernel K . To construct a Graph-Laplacian type kernel, we set $\mathbf{v}_i = \sqrt{\mathbf{W}^i} \mathbf{1}$, where \mathbf{W}^i is the i th row of \mathbf{W} , and define

$$k_{ij} = \frac{w_{ij}}{v_i v_j}.$$

To construct a Laplace-Beltrami type kernel, we set $\mathbf{v}_i = \mathbf{W}^i \mathbf{1}$ and compute

$$\tilde{k}_{ij} = \frac{w_{ij}}{v_i v_j}.$$

Then set $u_i = \sqrt{\tilde{K}^i \mathbf{1}}$, and define

$$k_{ij} = \frac{\tilde{k}_{ij}}{u_i u_j}.$$

Step 4. DR kernel decomposition. Let $\mathbf{v}^0, \mathbf{v}^1, \mathbf{v}^2, \dots, \mathbf{v}^d$ be the eigenvectors of \mathbf{K} achieving the $(d + 1)$ largest eigenvalues $1 = \lambda_0 > \lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d > 0$. Scale \mathbf{v}_i to

$$\tilde{\mathbf{v}}_i = \left[\frac{v_{1i}}{v_{10}}, \dots, \frac{v_{ni}}{v_{n0}} \right], \quad 1 \leq i \leq d.$$

Then the DR data matrix $\mathbf{Y} = [\tilde{\mathbf{v}}_1, \dots, \tilde{\mathbf{v}}_d]'$.

Remark 14.1. The neighborhood definition for Dmaps can be also realized by thresholding the weights appropriately. If the weight threshold method is used for the definition of neighborhood, Step 1 and Step 2 are merged to the following single step. Let τ be a weight threshold and $g_{ij} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{t}\right)$. Then the weight matrix $\mathbf{W} = [w_{ij}]_{ij=1}^n$ is constructed by

$$w_{ij} = \begin{cases} g_{ij}, & g_{ij} \leq \tau, \\ 0, & g_{ij} > \tau. \end{cases}$$

14.2.2 Dmaps Algorithm of Graph-Laplacian Type

The Matlab code of Dmaps algorithms is presented in the following. The first one adopts the Graph-Laplacian type normalization.

```
function [Y, par, ev] = drgldmaps(X, d, options)
% Graph-Laplacian type diffusion maps DR.
% SYNTAX:
%   Y = DRGLDMAPS(X, d, options)
% INPUTS:
%   X: D x n data matrix, X(:,i) is i-th point.
%   d: Dimension of the output data.
% OUTPUTS:
%   .epsilon: epsilon-neighbors.
%   .k: k-neighbors.
%   .par: Use parametric weights model
%         with parameter par (= -4t).
%   .scalingY: Scale output Y .
%   .verb: display the processing verbally.
% OUTPUT:
%   Y: d x n dimension-reduced data matrix.
```

```

% par: parameter in weight matrix (par =4t).
% Example: Reduce Swiss roll to 2-D.
% N=1000; D=2; SIGMA=1;
% tt = (3*pi/2)*(1+2*rand(1,N));
% height = 21*rand(1,N);
% X = [tt.*cos(tt); height; tt.*sin(tt)];
% Y = DRGLDMAPS(X,D);
%
% CALL: data_neighborhood
% Reference: S. Lafon, Ph.D Thesis (2004)
% Diffusion maps and geometric harmonics,
% Yale University.
% Code is written by Jianzhong Wang, Jan. 2008
%
% Initialize options.
n = size(X,2);
options.null=0;
if ~isfield(options, 'par') || isempty(options.par);
    parmode = false;
else
    parmode = true;
    par = options.par;
end
if ~isfield(options, 'scalingY')
    scalingY= false;
else
    scalingY= options.scalingY;
end
if isfield(options, 'verb')
    verb = option.berb;
else
    verb = 1;
end
% Step 1: Construct data-graph.
if verb
    fprintf(1,'--Construct data graph\n');
end
options.symmetric=1;
% X = X-min(X(:));
% X = X./max(X(:));
[D2, A] = data_neighborhood(X,options);
% Step 2: Construct weight matrix.
if verb
    fprintf(1,'--Construct weight matrix\n');
end
if ~parmode
    par = 4*mean(D2(A));
end
if par~=Inf
    spdq = -1/par*D2;

```

```

W = speye(n,n) + spfun(@exp,spdq);
clear spdq
else
    W = speye(n,n)+ double(A);
end
% Step 3: Construct DR kernel by G-L normalization.
if verb
    fprintf(1,'--Construct G-L Dmaps kernel.\n');
end
spvsm = sqrt(sum(W,2));
[r, c, v] = find(W);
clear W
v = v./(spvsm(r).*spvsm(c));
clear spvsm
gk = sparse(r, c, v, n, n);
% Step 4: Eigen decomposition
if verb
    fprintf(1,'--Find DR data.\n');
end
options.disp = 0; options.isreal = 1;
options.issym = 1;
[Y, ev] = eigs(gk, d+1,'LM',options);
Y = Y(:, 2:d+1);
if scalingY
    % re-scale the output.
    Y = Y./repmat(Y(:,1),1,d);
end
Y=Y';
D = diag(ev);
ev = sqrt(D(2:d+1));

```

14.2.3 Dmaps Algorithm of Laplace-Beltrami Type

The code of Dmaps using Laplace-Beltrami type normalization is the following.

```

function [Y, par, ev] = drlbdmaps(X,d,options)
% Laplace-Beltrami type diffusion maps for DR.
% SYNTAX:
%   Y = DRGLDMAPS(X,d,options)
% INPUTS:
%   X: D x n data matrix, X(:,i) is the i-th point.
%   d: Dimension of the output data.
% OUTPUTS:
%   .epsilon: epsilon-neighbors.
%   .k: k-neighbors.
%   .par: Use parametric weights model

```

```

%           with parameter par (= -4t).
% .scalingY: Scale output Y .
% .verb: display the processing verbally.
% OUTPUT:
%   Y: d x n dimension-reduced data matrix.
%   par: parameter in weight matrix (par =4t).
% Example: Reduce Swiss roll to 2-D.
% N=1000; D=2; SIGMA=1;
% tt = (3*pi/2)*(1+2*rand(1,N));
% height = 21*rand(1,N);
% X = [tt.*cos(tt); height; tt.*sin(tt)];
% Y = DRLBDMAPS(X,D);
%
% CALL: data_neighborhood
% Reference: S. Lafon, Ph.D Thesis (2004)
% Diffusion maps and geometric harmonics,
% Yale University.
% Code is written by Jianzhong Wang, Jan. 2008
%
% Initialize options.
n = size(X,2);
options.null=0;
if ~isfield(options,'par') || isempty(options.par)
    parmode = false;
else
    parmode = true;
    par = options.par;
end
if ~isfield(options, 'scalingY')
    scalingY= false;
else
    scalingY= options.scalingY;
end
if isfield(options, 'verb')
    verb = option.berb;
else
    verb = 1;
end
% Step 1: Construct data-graph.
if verb
    fprintf(1,'--Construct data graph\n');
end
options.symmetric=1;
[D2, A] = data_neighborhood(X,options);
% Step 2: Construct weight matrix.
if verb
    fprintf(1,'--Construct weight matrix\n');
end
if ~parmode
    par = 4*mean(D2(A));

```

```

end
if par~=Inf
    spdq = -1/par*D2;
    W = speye(n) + spfun(@exp,spdq);
    clear spdq
else
    W = speye(n)+ double(A);
end
% Step 3: Construct DR kernel by L-B normalization.
if verb
    fprintf(1,'--Construct L-B Dmaps kernel.\n');
end
spvsm = sum(W,2);
[r, c, v] = find(W);
v = v./(spvsm(r).*spvsm(c));
W = sparse(r, c, v, n, n);
spvsm = sqrt(sum(W,2));
[r, c, v] = find(W);
v = v./(spvsm(r).*spvsm(c));
gk = sparse(r, c, v, n, n);
% Step 4: Eigen decomposition
if verb
    fprintf(1,'--Find DR data.\n');
end
options.disp = 0; options.isreal = 1;
options.issym = 1;
[Y, ev] = eigs(gk, d+1,'LM',options);
if scalingY
    % re-scale the output.
    Y = Y./repmat(Y(:,1),1,d+1);
end
Y=(Y(:,2:d+1))';
D = diag(ev);
ev = sqrt(D(2:d+1));

```

Although the DR results of these two algorithms are similar, the difference between them exists. The implementation of these algorithms is presented in Section 14.4.

14.2.4 Dmaps Algorithm of Self-tuning Type

The scaling parameter t in the weight matrix impacts the accuracy of DR since it dominates the measurement of data similarities. Moreover, when the input data have different local statistics, perhaps there is no such a t that well defines the similarities for all of neighborhoods of the data.

Zelnik-Manor and Perona [13] proposed a variable local scaling parameter t_i in the computation of the weight w_{ij} :

$$w_{ij} = \exp\left(-\frac{-\|\mathbf{x}_j - \mathbf{x}_i\|^2}{t_i t_j}\right).$$

Using a specific scaling parameter for each point allows self-tuning of the point-to-point distances according to the local statistics of the neighborhoods surrounding points \mathbf{x}_i and \mathbf{x}_j . The selection of the local scale t_i can be made by studying the local statistics of the neighborhood of point \mathbf{x}_i . Suggested in [13], for a fixed m , t_i is computed by

$$t_i = d_2(\mathbf{x}_i, \mathbf{x}_{i_m}),$$

where \mathbf{x}_{i_m} is the m th neighbor of \mathbf{x}_i . The number m may be dependent on the dimension of the embedding space. The experiments in [13] show that m can be chosen in a wide range, say, from 5 to 20, and $m = 7$ is used in the implementation of [13].

The Dmaps algorithm of self-tuning can be created by changing the single scaling parameter t in the Dmaps algorithms above to a variable scaling parameter t_i . We give its Matlab code in the following.

```
function [Y,xm, ev] = drstdmaps(X, d, options)
% Self-tuning diffusion maps for DR.
% SYNTAX:
% [Y,xm] = DRSTDMAFS(X, d, options);
% INPUTS:
% X: D x n data matrix, X(:,i) is i-th point.
% d: Dimension of the output data.
% OUTPUTS:
% .epsilon: epsilon-neighbors.
% .k: k-neighbors.
% .tuning: the neighbor label for tuning.
% .par: Used for universal weight scaling
%       in (-4*par* t_i), usually is 1.
% .scalingY: True for scaling output Y.
% .GLtype: true for G-L normalization.
%           false for L-B normalization.
% .verb: display the processing verbally.
% OUTPUTS:
% Y: d x n dimension-reduced data matrix.
% xm: n-vector for the tuning distances.
% Example:
% options.PAR=1; options.TUNING=12;
% optopns.K=10; N=1000; D=2;
% tt = (3*pi/2)*(1+2*rand(1,N));
% height = 21*rand(1,N);
% X = [tt.*cos(tt); height; tt.*sin(tt)];
% Y = DRSTDMAFS(X,D,OPTIONS);
%
% CALL: data_neighborhood
%
% REFERENCE for self-tuning:
```

```
% Lihui Zelnik-Manor and P. Perona, (2004)
% "Self-tuning spectral clustering", Eighteenth
% Annual Conference on Neural Information
% Processing Systems, (NIPS)
%
% This code is created by Jianzhong Wang, 2008.
%
% Initialize inputs.
n = size(X,2);
options.null=0;
if isfield(options,'GLtype')
    GLtype = options.GLtype;
else
    GLtype = true;
end
if ~isfield(options,'par') || isempty(options.par)
    par = 1;
else
    par = options.par;
end
if ~isfield(options,'tuning')
    tuning_m = 7;
else
    tuning_m= options.tuning;
end
if ~isfield(options, 'scalingY')
    scalingY= false;
else
    scalingY= options.scalingY;
end
if isfield(options, 'verb')
    verb = option.verb;
else
    verb = 1;
end
% Step 1: Construct data-graph.
if verb
    fprintf(1,'--Construct data graph\n');
end
options.symmetric=1;
[D2, A] = data_neighborhood(X,options);
% Step 2: Distance tuning.
if verb
    fprintf(1,'--Tune local distance \n');
end
if isfield(options, 'k')
    k = options.k;
    if numel(k)>1
        k = min(k(:));
    end
```

```

else
    k = sum(A,2);
    k = min(k(:));
end
m = min(tuning_m,k);
sortD = sort(D2,'descend');
xm = sqrt(sortD(m,:));
D2 = D2./(xm'*xm);
% Step 3: Construct Weight matrix.
if verb
    fprintf(1,'--Construct weight matrix\n');
end
W = speye(n)+spfun(@exp,-D2/par);
% Step 4: Construct S-T Dmaps DR kernel.
if verb
    fprintf(1,'--Construct S-T Dmaps DR kernel\n');
end
if ~GLtype
    spvsm = sum(W,2);
    [r, c, v] = find(W);
    v = v./(spvsm(r).*spvsm(c));
    W = sparse(r, c, v, n, n);
end
spvsm = sqrt(sum(W,2));
[r, c, v] = find(W);
v = v./(spvsm(r).*spvsm(c));
gk = sparse(r, c, v, n, n);
% Step 5: Eigen decomposition.
if verb
    fprintf(1,'--Find DR data.\n');
end
options.disp = 0; options.isreal = 1;
options.issym = 1;
[Y, ev] = eigs(gk, d+1,'LM',options);
if scalingY
    % re-scale the output.
    Y = Y./repmat(Y(:,1),1,d+1);
end
Y=(Y(:,2:d+1))';
D = diag(ev);
ev = sqrt(D(2:d+1));

```

14.3 Implementation of Dmaps for DR

In this section, we demonstrate the Dmaps method in the dimensionality reduction for artificial surfaces. For comparing with other nonlinear DR methods, we adopt the same data settings as in LLE, LTSA, and Leigs.

14.3.1 Implementation of Dmaps of Graph-Laplacian Type

We first test Dmaps method of Graph-Laplacian type on artificial surfaces. For convenience, we abbreviate Dmaps method of Graph-Laplacian to GL-Dmaps, Dmaps method of Laplace-Beltrami type to LB-Dmaps, and Self-tuning Dmaps to ST-Dmaps. In Fig. 14.1, the 3-dimensional data of 4 artificial surfaces, 3D-cluster, Punched sphere, Swiss roll, and S-curve are reduced to 2-dimensional ones by GL-Dmaps algorithm. In the experiment, 1000 points are randomly sampled on each surface and 10 nearest points are used to construct the neighborhood of a point. The experiment shows that GL-Dmaps algorithm works well for S-curve, 3D-cluster, and Punched

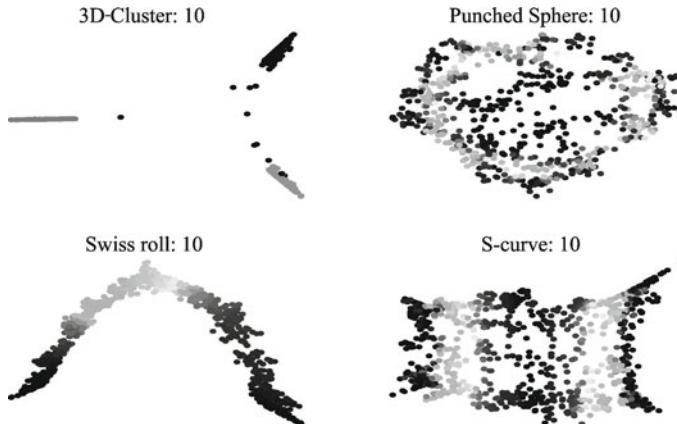


Fig. 14.1 GL-Dmaps algorithm is tested on 4 surfaces: 3D-cluster, Punched sphere, Swiss roll, and S-curve. 1000 points are randomly sampled on each of them. All are reduced to 2-dimensional data (on the plane). The original 3-dimensional graphs of these surfaces are not presented here. They can be found in Section 1.5. The figure displays the DR results for these surfaces. The graph shows that GL-Dmaps algorithm works well for them.

sphere. A certain shape distortion occurs on Swiss roll. Note that all Dmaps methods are based on the diffusion processing. Hence, it is not surprise that it can work well for nonsmooth surface such as 3D-cluster. Dmaps methods do not preserve the local Euclidean distances very well. Hence, the DR result of GL-Dmaps for Swiss roll does not keep the rectangular shape although there is no misclassification on the DR data set. In Fig. 14.2 and Fig. 14.3, we apply GL-Dmaps algorithm to reduce the dimension of noisy data, with the noise standard deviation 0.2 and 0.4 respectively. The results show that GL-Dmaps algorithm is insensitive to noise. This is due to the diffusion processing in Dmaps methods, which reduces the noise on the data. In Fig. 14.4, the different heights of Punched sphere are chosen for the test. It shows the results similar to those obtained by other methods: when the

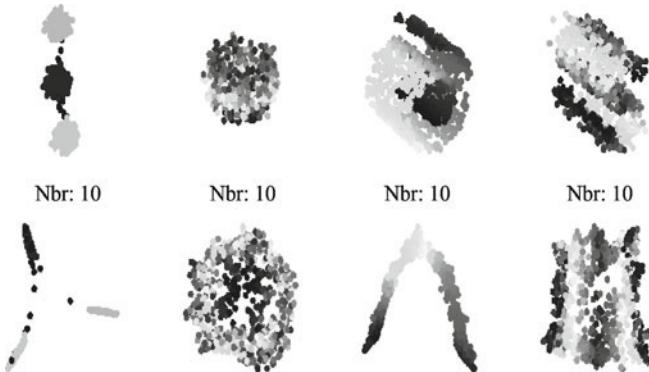


Fig. 14.2 GL-Dmaps algorithm is applied on 4 surfaces, with the noise standard deviation 0.2. All of the surfaces are sampled in the same way as in Fig. 14.1. The figure shows that GL-Dmaps algorithm is insensitive to noise. At the top line, four noisy data are presented. Top left: 3D-cluster, Top second: Punched sphere, Top third, Swiss roll, Top right: S-curve. At the bottom line are their corresponding dimensional reduced 2-dimensional data.

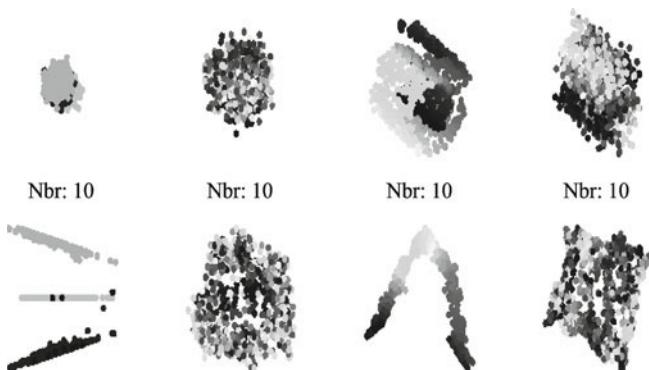


Fig. 14.3 GL-Dmaps algorithm is applied on four surfaces as in Fig. 14.2, with the noise standard deviation 0.4. The surfaces are sampled in the same way as in Fig. 14.1. The figure shows that when the noise deviation is large, the DR results of GL-Dmaps algorithm have certain distortions. At the top line, four noisy data are presented. Top left: 3D-cluster, Top second: Punched sphere, Top third, Swiss roll, Top right: S-curve. At the bottom line are their corresponding dimensional reduced 2-dimensional data.

height increases, the DR results have more distortions. Compared to others, GL-Dmaps method is relatively insensitive to the height change. In Fig. 14.5 GL-Dmaps algorithm is applied to the lengths of S-curve, 5, 10, and 30, with the same data settings as in Fig. 14.1. It is shown that when the length of the S-curve surface is too long, the distortion of the DR set occurs mainly due to the neighborhood inconsistence. Dmaps algorithms are insensitive to the neighborhood sizes. The algorithms remain stable over a wide range of

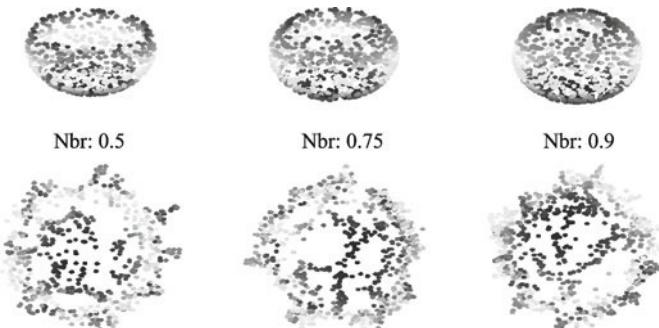


Fig. 14.4 Effect of surface shape on GL-Dmaps. The dimensionality reduction of the punched spheres is produced by GL-Dmaps for heights of the punched sphere, 0.5, 0.75, and 0.9, respectively, where the diameter of the original sphere is unit. It shows that when the height increases, more color mixture areas occur in the dimensionality reduction data.

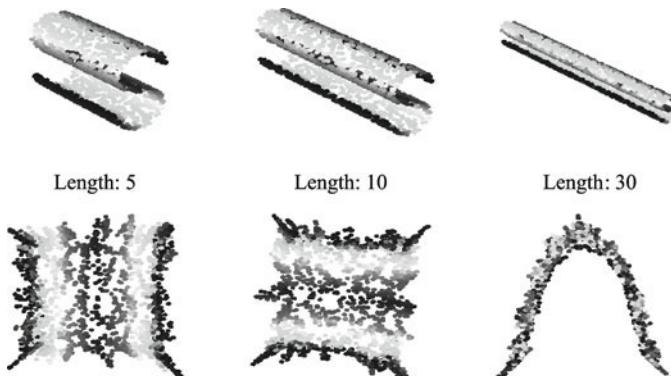


Fig. 14.5 The figure illustrates again that when the graph neighborhood is inconsistent with manifold, the DR data may distort the geometric structure of dat so that it becomes invalid. In this test, embeddings of the S-curves are computed by GL-Dmaps algorithm for different choices of the lengths, 5, 10, and 30, respectively. The surface data are sampled in the same way as in Fig. 14.1. The results show that the dimensionality reduction data are invalid when the length of the S-curve surface is 30. The distortion is due to the neighborhood inconsistence.

neighborhood sizes. Figure 14.6 displays the results of GL-Dmaps for the DR of S-curve data, showing that the algorithm is very stable. In the last experiment, the impact of the parameter t (see (14.7)) on the Dmaps DR kernel construction is tested. The results show that the algorithm is insensitive to the parameter, as shown in Fig. 14.7.

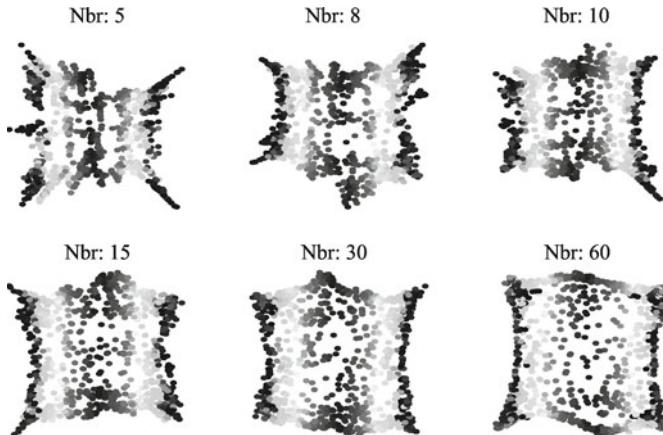


Fig. 14.6 Effect of neighborhood size on GL-Dmaps. The results show that GL-Dmaps algorithm is insensitive to the neighborhood size. In this figure, GL-Dmaps algorithm is applied to S-curve for different choices of the number of nearest neighbors k , 5, 8, 10, 15, 30, 60, respectively. A reliable GL-Dmaps embedding from 3dimension to 2dimension can be obtained over a wide range of values.

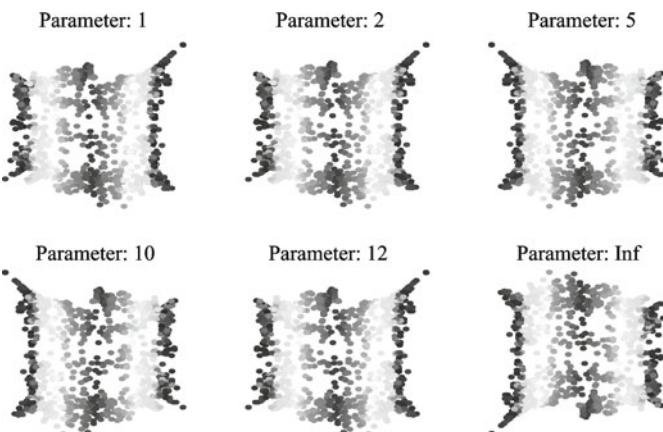


Fig. 14.7 Impact of the parameter t on GL-Dmaps algorithm. The results show that GL-Dmaps algorithm is insensitive to the value of the parameter t that is used in the kernel construction. In this figure, GL-Dmaps algorithm is applied to S-curve for different choices of the parameter in Gaussian, say 1, 2, 5, 10, 12, Inf, respectively. A reliable GL-Dmaps embedding from 3dimension to 2dimension can be obtained over a wide range of t values.

14.3.2 Implementation of Dmaps of Laplace-Beltrami Type

There are no significant differences between LB-Dmaps and GL-Dmaps. In [9], the author pointed out that the GL-Dmaps embedding is sensitive to the density of the points. Particularly, when the density has a steep peak, this embedding tends to map all points around this peak to a single point, creating a corner. Compared with GL-Dmaps, LB-maps is less sensitive to the point density (see Figure 2.5 in [9]).

In Fig. 14.8, the 3-dimensional data of 4 artificial surfaces, 3D-cluster, Punched sphere, Swiss roll, and S-curve, are reduced to 2-dimensional ones by LB-Dmaps algorithm. In the experiment, 1000 points are randomly sampled on each surface and 10 nearest points are used to construct the neighborhood of a point. The experiment shows that LB-Dmaps algorithm produces the

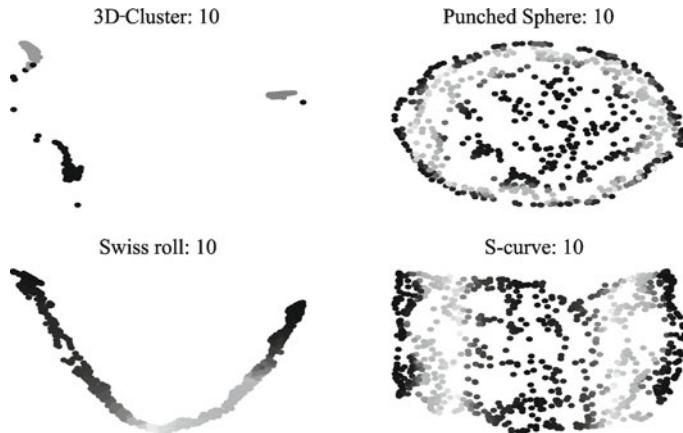


Fig. 14.8 LB-Dmaps algorithm is tested on 4 surfaces: 3D-cluster, Punched sphere, Swiss roll, and S-curve. 1000 points are randomly sampled on each of them. All are reduced to 2-dimensional data (on the plane). The figure displays the DR results for these surfaces. The graphs show that LB-Dmaps algorithm produces less *peak* points than GL-Dmaps, particularly for the DR data for Punched sphere and S-curve.

results similar to GL-Dmaps, but less sensitive to the density of the points. For example, the DR results for Punched sphere and S-curve of LB-Dmaps contain less shape distortion than GL-maps. In Fig. 14.9, the different heights of Punched sphere are chosen for the test of LB-Dmaps embedding. It shows the results similar to those obtained by GL-Dmaps: LB-Dmaps method is relatively insensitive to the height change. Again we can see that the DR data of LB-Dmaps contain less irregular points than GL-maps. In the last two experiments, the impact of the neighbor size and the parameter t on the LB-Dmaps embedding are displayed. The results show again that the algorithm is insensitive to either the neighbor size or the parameter t , and

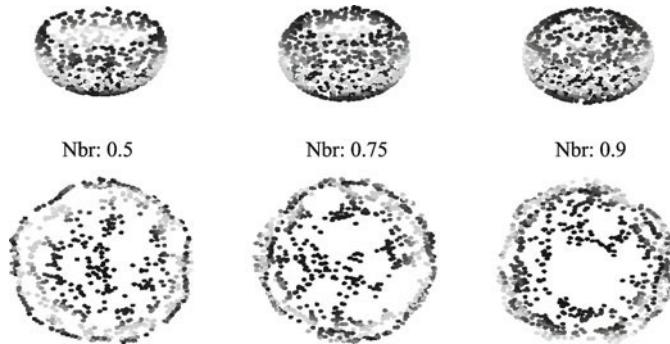


Fig. 14.9 Effect of surface shape on LB-Dmaps. The dimensionality reduction of the punched spheres is produced by LB-Dmaps for heights of the punched sphere, 0.5, 0.75, and 0.9, respectively, where the diameter of the original sphere is unit. It shows that the DR data of LB-Dmaps contain less irregular points than GL-maps.

the DR data set has less irregular points, as shown in Fig. 14.10 and Fig. 14.11, respectively.

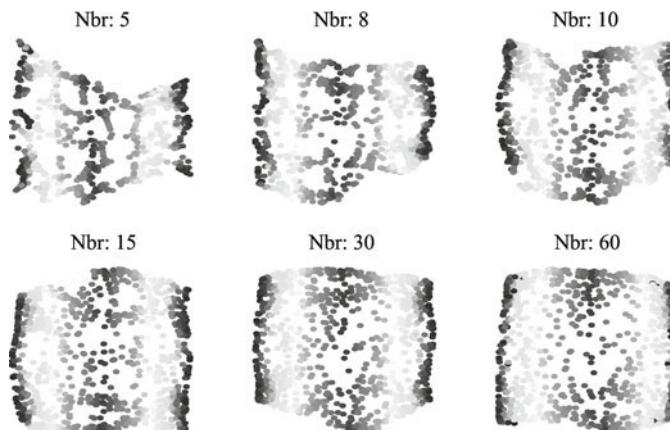


Fig. 14.10 Impact of neighborhood size on LB-Dmaps. The results show that LB-Dmaps algorithm is insensitive to the neighborhood size. It is also shown that the DR data produced by LB-Dmaps have less irregular points than GL-Dmaps.

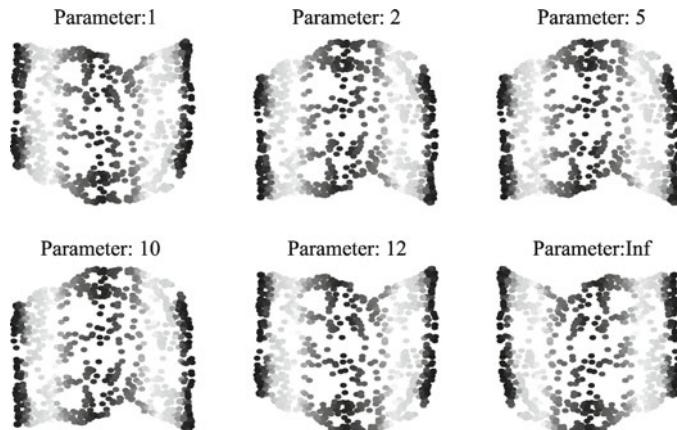


Fig. 14.11 Impact of the parameter t on LB-Dmaps algorithm. The results show that BL-Dmaps algorithm is insensitive to the value of the parameter t . It is also shown that the DR data have less irregular points than GL-Dmaps.

14.3.3 Implementation of Dmaps of Self-turning Type

There are no significant differences between ST-Dmaps and GL-Dmaps. In Fig. 14.12, the 3-dimensional data of 4 artificial surfaces, 3D-cluster, Punched

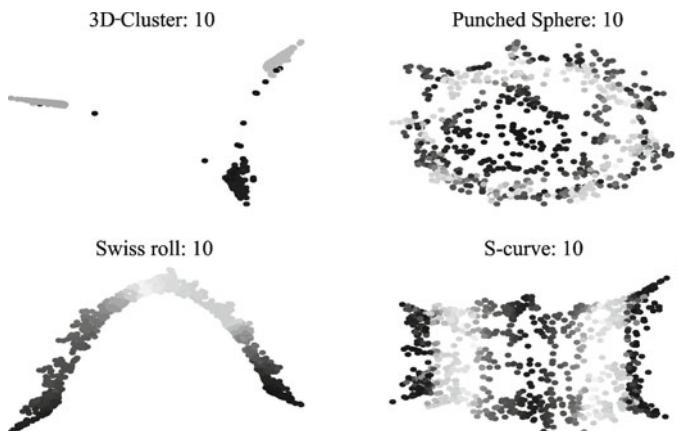


Fig. 14.12 ST-Dmaps algorithm is tested on 4 surfaces: 3D-cluster, Punched sphere, Swiss roll, and S-curve. 1000 points are randomly sampled on each of them. All are reduced to 2-dimensional data (on the plane). The figure displays the DR results for these surfaces. The graphs show that ST-Dmaps algorithm produces results very similar to GL-Dmaps.

sphere, Swiss roll, and S-curve, are reduced to 2-dimensional ones by ST-Dmaps algorithm. In the experiment, 1000 points are randomly sampled on each surface and the tuning number m is set at 10. The experiment shows that ST-Dmaps algorithm produces the results similar to those of GL-Dmaps. If we apply the same settings to other tests used above, the DR results of ST-Dmaps are also very similar to those obtained by GL-Dmaps embedding. Hence, we skip the figures for those tests.

14.3.4 Conclusion

The construction of Dmaps kernels is very simple. The neighborhood construction for the diffusion maps method can be obtained from either k -neighborhood or ε -neighborhood, and even can be obtained by thresholding the weights appropriately. In addition, by taking advantage of the semi-group structure, the fast, stable diffusion wavelet algorithms developed in [14] can be used to accelerate the computation of eigen decomposition of the DR kernel. Since Dmaps algorithms adopt diffusion processing, they are less sensitive to noise. The experiments show that they are also insensitive to the parameter t in the weight matrix. Dmaps methods do not exactly preserve the local distances. Hence, the shape of the DR data set sometimes are not preserved well.

14.4 Diffusion Maps and Multiscale Diffusion Geometry

14.4.1 Construction of General Diffusion Kernels

To reveal the geometric relation between the original data set \mathcal{X} and the DR data set \mathcal{Y} produced by Dmaps, we study the diffusion maps in a more general framework.

Let $M \subset \mathbb{R}^D$ be a d -manifold on which the data set \mathcal{X} lies on. Suppose that the geometry of M is characterized by a kernel $k(\mathbf{x}, \mathbf{y})$, that is, the kernel $k(\mathbf{x}, \mathbf{y})$ measures the similarities between the points on M . For instance, the Dmaps method chooses the Gaussian $G_t(\mathbf{x}, \mathbf{y})$ to represent such a non-normalized kernel. If the kernel is well chosen, then it represents the proper information of M .

Definition 14.1. Let $M \subset \mathbb{R}^D$ be a d -manifold and μ be a finite measure on M so that $d\mu(\mathbf{x}) = p(\mathbf{x})d\mathbf{x}$, where $p(\mathbf{x})$ is a distribution density function on M and $d\mathbf{x}$ is the Riemannian measure. A kernel $k(\mathbf{x}, \mathbf{y})$ on M is called a diffusion kernel if it satisfies the following conditions:

1. k is symmetric: $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x})$.
2. k is positivity preserving: for all $\mathbf{x}, \mathbf{y} \in M$, $k(\mathbf{x}, \mathbf{y}) \geq 0$.
3. k is positive semi-definite: for all bounded functions $f \in L^2(M)$,

$$\int_M \int_M k(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) f(\mathbf{y}) d\mu(\mathbf{x}) d\mu(\mathbf{y}) \geq 0.$$

In the following, these conditions on k will be referred to as the admissible conditions. This type of kernel can be constructed by

$$k(\mathbf{x}, \mathbf{y}) = \eta(\|\mathbf{x} - \mathbf{y}\|).$$

When η is the Fourier transform of a positive measure, Bochner's Theorem [15] guarantees the positivity of the kernel. To normalize k , we apply the method used in Section 14.1. Define

$$v^2(\mathbf{x}) = \int_M k(\mathbf{x}, \mathbf{y}) d\mu(\mathbf{y})$$

and set $\tilde{a}(\mathbf{x}, \mathbf{y}) = \frac{k(\mathbf{x}, \mathbf{y})}{v^2(\mathbf{x})}$. Then, we have

$$\int_M \tilde{a}(\mathbf{x}, \mathbf{y}) d\mu(\mathbf{y}) = 1.$$

Note that the new kernel \tilde{a} is still positive $\tilde{a}(\mathbf{x}, \mathbf{y}) \geq 0$ and its corresponding operator

$$(\tilde{\mathcal{A}}f)(\mathbf{x}) = \int_M \tilde{a}(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mu(\mathbf{y})$$

is positivity-preserving, i.e., $\tilde{\mathcal{A}}f \geq 0$ if $f \geq 0$. The kernel \tilde{a} is not symmetric. To make a symmetric kernel, we employ

$$a(\mathbf{x}, \mathbf{y}) = \frac{k(\mathbf{x}, \mathbf{y})}{v(\mathbf{x})v(\mathbf{y})} = \frac{v(\mathbf{x})}{v(\mathbf{y})} \tilde{a}(\mathbf{x}, \mathbf{y})$$

and

$$(\mathcal{A}f)(\mathbf{x}) = \int_M a(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mu(\mathbf{y}).$$

Then the kernel a is a positive, symmetric, semi-definite kernel, and has norm 1, so that the norm of the operator \mathcal{A} on $L^2(M)$ is 1:

$$\|\mathcal{A}\| = 1. \tag{14.8}$$

It is obvious that v is the eigenfunction achieving 1:

$$\mathcal{A}v = v. \tag{14.9}$$

To prove (14.8) and (14.9), we apply the Cauchy-Schwartz inequality, obtaining

$$\begin{aligned} & \left| \int_M k(\mathbf{x}, \mathbf{y}) \frac{f(\mathbf{y})}{v(\mathbf{y})} d\mu(\mathbf{y}) \right| \\ & \leq \left(\int_M k(\mathbf{x}, \mathbf{y}) d\mu(\mathbf{y}) \right)^{1/2} \left(\int_M k(\mathbf{x}, \mathbf{y}) \frac{f^2(\mathbf{y})}{v^2(\mathbf{y})} d\mu(\mathbf{y}) \right)^{1/2} \\ & = v(\mathbf{x}) \left(\int_M k(\mathbf{x}, \mathbf{y}) \frac{f^2(\mathbf{y})}{v^2(\mathbf{y})} d\mu(\mathbf{y}) \right)^{1/2}. \end{aligned}$$

Consequently,

$$\begin{aligned} \langle \mathcal{A}f, f \rangle &= \int_M \int_M k(\mathbf{x}, \mathbf{y}) \frac{f(\mathbf{x})f(\mathbf{y})}{v(\mathbf{x})v(\mathbf{y})} d\mu(\mathbf{x})d\mu(\mathbf{y}) \\ &\leq \int_M |f(\mathbf{x})| d\mu(\mathbf{x}) \left(\int_M k(\mathbf{x}, \mathbf{y}) \frac{f^2(\mathbf{y})}{v^2(\mathbf{y})} d\mu(\mathbf{y}) \right)^{1/2} \\ &\leq \|f\|^2. \end{aligned}$$

Hence, $\|\mathcal{A}\| \leq 1$. The direct computations shows that $\mathcal{A}v = v$, which derives $\|\mathcal{A}\| = 1$.

14.4.2 Diffusion Distances

The operator \mathcal{A} is positive, self-adjoint, and has norm 1. Hence, the spectral decomposition of the kernel a is

$$a(\mathbf{x}, \mathbf{y}) = \sum_{j=0}^{\infty} \lambda_j \phi_j(\mathbf{x}) \phi_j(\mathbf{y}),$$

where the eigenvalues are assumed to be descended:

$$1 = \lambda_0 \geq \lambda_1 \geq \dots \lambda_d \geq \dots \rightarrow 0.$$

Let $a^{(m)}(\mathbf{x}, \mathbf{y})$ denote the kernel of the power \mathcal{A}^m . Then

$$a^{(m)}(\mathbf{x}, \mathbf{y}) = \sum_{j=0}^{\infty} \lambda_j^m \phi_j(\mathbf{x}) \phi_j(\mathbf{y}), \quad (14.10)$$

which corresponds to the diffusion operator

$$\mathcal{A}^m(f)(\mathbf{x}) = \int_M a^{(m)}(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mu(\mathbf{y}). \quad (14.11)$$

If we consider f as a state function on M , then the equation (14.11) interprets the probability for a Markov chain with the transition matrix a to reach \mathbf{x} from \mathbf{y} in m steps.

All of the eigenfunctions ϕ_j are feature functions of M , characterizing the geometry of M while the eigenvalues measure the degrees of significance of these feature functions. Since $0 \leq \lambda_j \leq 1$, as m increases, only a few feature functions survive in the sum (14.10), ignoring those ϕ_j with λ_j^m less than a certain threshold. This means that to represent the significant features of M , only a small number of eigenfunctions are needed. Treating these eigenfunctions as coordinates, we can represent M using only these coordinates, reducing the data on M to a small dimensionality. The space spanned by these eigenfunctions is called the feature space. For a given threshold, the dimension of the feature space is dependent on the scale m : The larger the integer m , the smaller the dimension of the feature space. We shall say that the semi-group $\{\mathcal{A}^m\}_{m=0}^\infty$ of diffusion operators represents a multiscale diffusion geometry of M associated with the feature functions $\phi_j, j \geq 0$.

The coordinate representation of $\mathbf{x} \in M$ provided by eigenfunctions defines the diffusion maps.

Definition 14.2. Let $a(\mathbf{x}, \mathbf{y})$ be a given diffusion kernel, which has a spectral decomposition (14.10). Then the mapping from M to l^2 :

$$\Phi(s) = \begin{pmatrix} \phi_0(\mathbf{x}) \\ \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \\ \vdots \end{pmatrix}$$

is called a feature mapping, and $\phi_j(\mathbf{x})$ is interpreted as the j th coordinate of \mathbf{x} under the feature basis $\{\phi_j\}_{j=0}^\infty$.

The feature coordinate representation of \mathbf{x} is often more meaningful than its Euclidean representation $\mathbf{x} = [x_1, \dots, x_D] \in \mathbb{R}^D$.

When each point is represented by the feature coordinates, it is natural to measure the distance between two points \mathbf{x} and \mathbf{y} in the feature space. Therefore, we give the following definition.

Definition 14.3. Let $a(\mathbf{x}, \mathbf{y})$ be a given diffusion kernel, which has a spectral decomposition (14.10). Then the diffusion metric (at level m) on M is defined by

$$\|\mathbf{x}\|_{a,m} = \sqrt{\sum_{j=0}^{\infty} \lambda_j^m (\phi_j(\mathbf{x}))^2}$$

and the diffusion distance between \mathbf{x} and \mathbf{y} (at level m) is defined by

$$D_m(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{j=0}^{\infty} \lambda_j^m (\phi_j(\mathbf{x}) - \phi_j(\mathbf{y}))^2}. \quad (14.12)$$

Let the weighted Euclidean distance in l^2 with the weights $\{w_i\}$ be defined by

$$\|\mathbf{u} - \mathbf{v}\|_w = \sqrt{\sum_i w_i (u_i - v_i)^2}.$$

Hence, the diffusion distance is a weighted Euclidean distance in the feature space. Numerically, for a given threshold $\delta > 0$, we can approximately write

$$D_m^2(\mathbf{x}, \mathbf{y}) = \sum_{j=0}^d \lambda_j^m (\phi_j(\mathbf{x}) - \phi_j(\mathbf{y}))^2, \quad \lambda_{d+1}^m < \delta \leq \lambda_d^m. \quad (14.13)$$

If λ_j exponentially decays, the approximation above will not lose much energy when m is large. Hence, we shall call the mapping

$$\Phi_d(\mathbf{x}) = \begin{pmatrix} \phi_0(\mathbf{x}) \\ \phi_1(\mathbf{x}) \\ \vdots \\ \phi_d(\mathbf{x}) \end{pmatrix}$$

diffusion map. Recall that the kernel $a^{(m)}$ is a diffusion kernel, which is positive semi-definite. Therefore, the diffusion distance $D_m(\mathbf{x}, \mathbf{y})$ has the following kernel representation.

Lemma 14.1. *We have the following.*

$$D_m^2(\mathbf{x}, \mathbf{y}) = a^{(m)}(\mathbf{x}, \mathbf{x}) + a^{(m)}(\mathbf{y}, \mathbf{y}) - 2a^{(m)}(\mathbf{x}, \mathbf{y}).$$

Proof. By the spectral decomposition of $a^{(m)}$ in (14.10), we have

$$\begin{aligned} a^{(m)}(\mathbf{x}, \mathbf{x}) &= \sum_{j=0}^{\infty} \lambda_j \phi_j^2(\mathbf{x}), \\ a^{(m)}(\mathbf{y}, \mathbf{y}) &= \sum_{j=0}^{\infty} \lambda_j \phi_j^2(\mathbf{y}), \\ a^{(m)}(\mathbf{x}, \mathbf{y}) &= \sum_{j=0}^{\infty} \lambda_j \phi_j(\mathbf{x}) \phi_j(\mathbf{y}), \end{aligned}$$

which yields

$$\begin{aligned} D_m^2(\mathbf{x}, \mathbf{y}) &= \sum_{j=0}^{\infty} \lambda_j (\phi_j(\mathbf{x}) - \phi_j(\mathbf{y}))^2 \\ &= \sum_{j=0}^{\infty} \lambda_j (\phi_j^2(\mathbf{x}) + \phi_j^2(\mathbf{y}) - 2\phi_j(\mathbf{x})\phi_j(\mathbf{y})) \\ &= a^{(m)}(\mathbf{x}, \mathbf{x}) + a^{(m)}(\mathbf{y}, \mathbf{y}) - 2a^{(m)}(\mathbf{x}, \mathbf{y}). \end{aligned}$$

The lemma is proved. \square

Finally, we give an interpretation of the 1-eigenfunction $v(\mathbf{x})$. Recall that if the (non-normalized) diffusion kernel is the Gaussian

$$G_t(\mathbf{x}, \mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{4tc}},$$

where c is the thermal diffusivity, then $v(\mathbf{x}) \approx (4\pi ct)^{d/4}$. In the isotropic diffusion $c = \frac{k}{c_p \rho}$, where k is the thermal conductivity, c_p is the specific heat capacity, and ρ is the mass density. Let k and c_p be fixed, then $v(\mathbf{x})$ is the reciprocal of the mass density. In information theory, $v(\mathbf{x})$ can be interpreted as the *density of data*.

The results of the continuous model we discussed above can be easily converted to the results of the discrete model. Assume that the data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ is given. Then the discrete form of the kernel $k(\mathbf{x}, \mathbf{y})$ is an $n \times n$ matrix whose (i, j) -entry can be represented as

$$k_{ij} = \eta(\|\mathbf{x}_i - \mathbf{x}_j\|),$$

where $\eta > 0$ is a decreasing function on $[0, \infty)$. The normalization step is very similar to what we have done in Section 14.1. We shall skip the details here.

14.4.3 Diffusion Maps as Feature Extractors

From the previous subsection, we see that diffusion maps can serve as feature extractors, presenting multiscale diffusion features of data. Dmaps method then extracts data features in dimension reduction processing. The reduced data have a twofold interpretation: Each column of the data matrix provides the feature coordinates of the objective vector in the original data, and each row represents a feature function on the data, arranged according to the degree of significance. Assume that the intrinsic dimension of the data is d . Then all eigenfunctions $\phi_j, j > d + 1$ in Definition 14.2 correspond to noise. When m increases, λ_j^m quickly tends to zero. Hence, for a certain m , only a few feature functions occur in the numerical sense. Several examples in the next section illustrate the ability of Dmaps in data feature extraction.

14.5 Implementation of Dmaps for Feature Extraction

14.5.1 Feature Extracted from 3-dimensional Toroidal Helix

In the first example, we illustrate that Dmaps method can be considered as a feature extractor for data sets. The original data is a toroidal helix in the space. Its 1-dimensional topology is a circle. The Dmaps method produces a very good embedding for the helix, showing its DR data as a circle on the plane. In Fig. 14.13, the helix data is reduced on the plane by GL-Dmaps and LB-Dmaps algorithms respectively. Both reduced data have a very good shape of circle.



Fig. 14.13 Left: the toroidal helix. Middle: the embedding of the helix obtained by using GL-Dmaps. Right: the embedding of the helix obtained by using LB-Dmaps.

14.5.2 Reordering Face Images

We studied a sequence of face images from the UMIST Face Database-1a in http://tutorial-haartraining.googlecode.com/svn/trunk/data/umist_cropped (see Section 5.3, also [9, 16]). There is a set of 36 faces of a same person turning his head. Each picture is a pre-cropped 112×92 pixels gray image, and the time sampling rate is sufficiently high (Fig. 14.14). They are expected to be organized along a curve. Dmaps method of Laplace-Beltrami type is applied as follows.

- Initially, the pictures are indexed by the time parameter, or equivalently, by the angle of the head. To illustrate the capability of reorganization of the method, the faces are randomly arranged so that they appear unordered.
- The Euclidean distances between the images in the set are measured. Then the eigenfunctions of the Laplace-Beltrami operator on this structure are computed. Finally, the pictures are ordered according to the

weights on the eigenfunctions.



Fig. 14.14 The face images in a training set are chosen from the UMIST Face Database-1a. The pictures are ordered from left to right, and top to bottom.

The test result is illustrated in Fig. 14.15. It shows that Dmaps method well orders the pictures.



Fig. 14.15 The left is the unordered pictures. The right is the ordered pictures obtained by Dmaps.

14.5.3 Image Parameters Revealing

In [9], the author gave an example to show that the Dmaps method well discovers the parameters used for image display. In the experiment, a database of images that are parameterized by two real numbers is displayed on a surface. More precisely, the set of images, say, *Gamma*, is composed of a sequence of 1275 images (75×81 pixels) of the word 3D viewed at different angles. Each image was generated by a renderer software from a three-dimensional model for the two letters 3 and D, and the object was rotated along the vertical axis (angle α) and horizontal axis (angle β), as shown in Fig. 14.16. The data were highly sampled: α was uniformly sampled from -50° to 50° with a step of 2° , whereas β was sampled once every 4° from -46° to 50° . From the data, a data graph is created so that each point is connected with its 8 nearest neighbors. Then each edge (x, y) was assigned the weight $e^{-\frac{\|x-y\|^2}{t}}$. The weights are normalized into Graph-Laplacian type. After applying the Dmaps method (of Graph-Laplacian type), the original images of the words 3D are mapped on the feature plane with the coordinates (ϕ_1, ϕ_2) (Fig. 14.17). The result shows that the orientation of the object can be controlled by the



Fig. 14.16 A part of the original picture set, which consists of 1275 images (75×81 pixels) of the word 3D. In the original set, the angle α is discretized 51 times between $-50^\circ \rightarrow 50^\circ$ in the horizontal direction and the angle β is discretized 25 times between $-50^\circ \rightarrow 50^\circ$ in the vertical direction. The graph is copied from the original in [9].

two coordinates ϕ_1 and ϕ_2 . What appears in Fig. 14.17 is that there is a bi-Lipschitz mapping between the angles (α, β) and the coordinates (ϕ_1, ϕ_2) . In other words, the natural parameters of the data set have been recovered by the Dmaps DR algorithm. The rearranged result obtained by Dmaps is illustrated in Fig. 14.17. It shows that Dmaps method well reveals Angles α and β . The eigenfunctions ϕ_1 and ϕ_2 correspond to α and β respectively.



Fig. 14.17 Some images of the picture set Γ are plotted in the feature plane with the coordinates (ϕ_1, ϕ_2) . The angle parameters α and β are recovered. The graph is copied from the original in [9].

14.5.4 Feature Images of Hyperspectral Image Cube

Dmaps can also be applied to constructing the feature images for the hyperspectral images [9]. In a hyperspectral image cube, there are about several hundreds band images. Applying Dmaps DR method, several feature images are extracted from the cube. They well represent the features of the whole image cube and therefore can be used in HSI data processing. Figure 14.18 demonstrates a hyperspectral pathology data cube, and in Fig. 14.19 the feature images of the data are presented. Dmaps can be applied to many

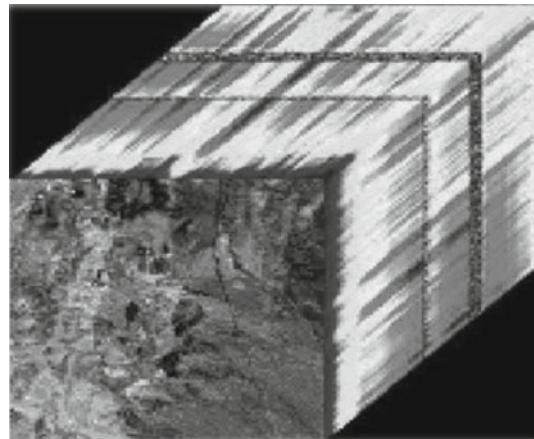


Fig. 14.18 Hyperspectral image data cube. It is modified from its original figure in Lafon's dissertation [9].

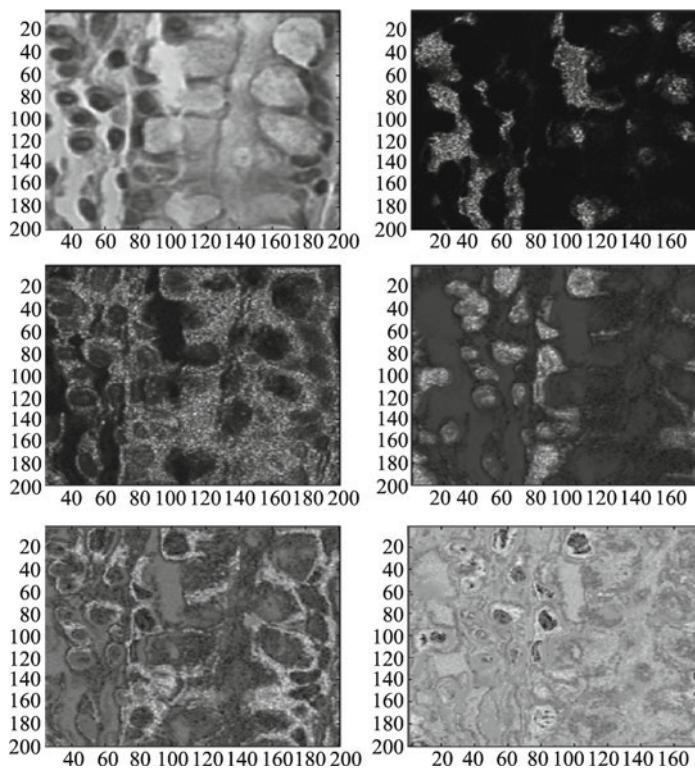


Fig. 14.19 From left to right, top to bottom: they are the eigenfunctions $\phi_k, k = 1, 2, \dots, 6$, of Dmaps kernel on the data set. It is modified from its original figure in Lafon's dissertation [9].

other applications. The readers may refer to the references of this chapter and the material on <http://research.google.com/pubs/papers.html>.

References

- [1] Kannan, R., Vempala, S., Vetta, V.: On spectral clustering – good, bad and spectral. In: Proceedings of the 41st Annual Symposium on Foundations of Computer Science (2000).
- [2] Meila, M., Shi, J.: Learning segmentation by random walks. In: Advances in Neural Information Processing Systems (2001).
- [3] Ng, A., Jordan, M., Weiss, Y.: On spectral clustering: Analysis and an algorithm. In: Advances in Neural Information Processing Systems, vol. 14 (2001).
- [4] Perona, P., Freeman, W.T.: A factorization approach to grouping. In: Proceedings of the 5th European Conference on Computer Vision, pp. 655–670 (1998).
- [5] Polito, M., Perona, P.: Grouping and dimensionality reduction by locally linear embedding. In: Advances in Neural Information Processing Systems, vol. 14 (2002).
- [6] Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Trans. Pattern Analysis and Machine Intelligence 22(8), 888–905 (2000).
- [7] Vapnik, V.: Statistical Learning Theory. Wiley-Interscience (1998).
- [8] Coifman, R.R., Lafon, S.: Diffusion maps. Appl. Comput. Harmon. Anal. 21, 5–30 (2006).
- [9] Lafon, S.: Diffusion maps and geometric harmonics. Ph.D. thesis, Yale University (2004).
- [10] Lafon, S., Keller, Y., Coifman, R.R.: Data fusion and multicue data matching by diffusion maps. IEEE Trans. Pattern Analysis and Machine Intelligence 28(11), 1784–1797 (2006).
- [11] Lafon, S., Lee, A.B.: Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and data set parameterization. IEEE Trans. Pattern Analysis and Machine Intelligence 28(9), 1393–1403 (2006).
- [12] Nadler, B., Lafon, S., Coifman, R.R., Kevrekidis, I.G.: Diffusion maps, spectral clustering and reaction coordinates of dynamical systems. Appl. Comput. Harmon. Anal. 21, 113–1127 (2006).
- [13] Zelnik-Manor, L., Perona, P.: Self-tuning spectral clustering. In: Eighteenth Annual Conference on Neural Information Processing Systems (NIPS) (2004).
- [14] Coifman, R.R., Maggioni, M.: Diffusion wavelets in Special Issue on Diffusion Maps and Wavelets. Appl. Comput. Harmon. Anal. 21, 53–94 (2006).
- [15] Reed, M., Simon, B.: Methods of Modern Mathematical Physics, vol. II. Academic Press (1975).
- [16] Graham, D.B., Allinson, N.M.: Characterizing virtual eigensignatures for general purpose face recognition. Computer and Systems Sciences 163, 446–456 (1998).

Chapter 15 Fast Algorithms for DR Approximation

Abstract In nonlinear dimensionality reduction, the kernel dimension is the square of the vector number in the data set. In many applications, the number of data vectors is very large. The spectral decomposition of a large dimensional kernel encounters difficulties in at least three aspects: large memory usage, high computational complexity, and computational instability. Although the kernels in some nonlinear DR methods are sparse matrices, which enable us to overcome the difficulties in memory usage and computational complexity partially, yet it is not clear if the instability issue can be settled. In this chapter, we study some fast algorithms that avoid the spectral decomposition of large dimensional kernels in DR processing, dramatically reducing memory usage and computational complexity, as well as increasing numerical stability. In Section 15.1, we introduce the concepts of rank revealings. In Section 15.2, we present the randomized low rank approximation algorithms. In Section 15.3, greedy rank-revealing algorithms (GAT) and randomized anisotropic transformation algorithms (RAT), which approximate leading eigenvalues and eigenvectors of DR kernels, are introduced. Numerical experiments are shown in Section 15.4 to illustrate the validity of these algorithms. The justification of RAT algorithms is included in Section 15.5.

15.1 Low-rank Approximation and Rank-revealing Factorization

In the PCA algorithm, the key step is to make the SVD decomposition of the data matrix (see Theorem 5.3 in Section 5.2). In nonlinear DR algorithms, the DR date is derived from the spectral decomposition of DR kernels. When the dimension of a DR kernel is very large, the spectral decomposition of the kernel is a challenging problem. Therefore, an efficient low-rank approximation of the decomposition is a ubiquitous task. In this section we introduce the concept of rank-revealing factorization, which directly leads to low-rank

approximation of matrices. In the context, we often use the term *matrix decomposition* for *matrix factorization*.

15.1.1 Rank-revealing Factorization

As an example of matrix decomposition, we review the SVD factorization. Assume that $\mathbf{A} \in \mathfrak{M}_{m,n}(k)$ is an $m \times n$ matrix with rank k . The SVD factorization of \mathbf{A} is

$$\mathbf{A} = \sum_{j=1}^k \mathbf{u}_j \sigma_j (\mathbf{v}_j)' = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}', \quad (15.1)$$

where $\mathcal{U} = \{\mathbf{u}_1, \dots, \mathbf{u}_k\}$ is an o.n. basis of the k -dimensional subspace spanned by the columns of \mathbf{A} , and $\mathcal{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ is the o.n. basis of the functionals on $\text{Col}(\mathbf{A})$. Let $d \leq k$. Write

$$\mathbf{A}_d = \sum_{j=1}^d \mathbf{u}_j \sigma_j (\mathbf{v}_j)' = \mathbf{U}_d \mathbf{D}_d \mathbf{V}'_d. \quad (15.2)$$

Then

$$\|\mathbf{A} - \mathbf{A}_d\| = \sigma_{d+1}, \quad \|\mathbf{A} - \mathbf{A}_d\|_{\text{F}}^2 = \sum_{j=d+1}^k \sigma_j^2.$$

Hence, \mathbf{A}_d is the best approximation of \mathbf{A} among all d -rank $m \times n$ matrices under the spectral norm (and the Frobenius norm as well). We call \mathbf{A}_d the best d low-rank approximation of \mathbf{A} , or the best d -rank-revealing approximation of \mathbf{A} . Since SVD is computationally expensive when m and n are large, people try to find some approximative algorithms that are fast but do not lose too much accuracy. This idea leads to:

Definition 15.1. Let $\mathbf{A} \in \mathfrak{M}_{m,n}(k)$ have singular values $\sigma_1 \geq \dots \geq \sigma_k > 0$ and $d \leq k$. A matrix $\tilde{\mathbf{A}}_d \in \mathfrak{M}_{m,n}(d)$ is called a d -rank approximation of \mathbf{A} if

$$\|\mathbf{A} - \tilde{\mathbf{A}}_d\| \leq c \sigma_{d+1}, \quad (15.3)$$

where $c \geq 1$ is a constant independent of \mathbf{A} .

When $c = 1$, $\tilde{\mathbf{A}}_d$ is uniquely given by \mathbf{A}_d in (15.2). The norm in (15.3) can also be replaced by the Frobenius norm as σ_{d+1} is replaced by $\sqrt{\sum_{j=d+1}^k \sigma_j^2}$. When $c > 1$, the approximation matrix $\tilde{\mathbf{A}}_d$ is not unique. The algorithm that realizes the low-rank approximation is called a low-rank approximation algorithm. For example, we have

Definition 15.2. An algorithm is called a d -rank approximation SVD algorithm, if, for each $\mathbf{A} \in \mathfrak{M}_{m,n}(k)$, the algorithm produces three matrices $\mathbf{V} \in \mathfrak{D}_{D,d}$, $\boldsymbol{\Sigma} \in \mathfrak{D}_d$, and $\mathbf{U} \in \mathfrak{D}_{n,d}$, such that

$$\|\mathbf{A} - \mathbf{V}\boldsymbol{\Sigma}\mathbf{U}'\| \leq c\sigma_{d+1}, \quad (15.4)$$

where $c \geq 1$ is a constant independent of \mathbf{X} .

Note that if the rank of \mathbf{A} is d , then d -rank approximation SVD algorithms produce the exact SVD decomposition of \mathbf{A} . When $c > 1$ and $\sigma_{d+1} \neq 0$, the algorithms do not provide the best approximation as given in (15.2). However, as a consequence of relaxing the constraint $c = 1$, we may develop algorithms that reduce computational complexity. In approximative algorithms, we often need to make trade-offs between computational complexity and precision. The use of approximate matrices appears promising, as the degradation in performance is often marginal compared to the gain in processing speed. Hence, low-rank approximation algorithms are very attractive in the applications, which are involved in large dimensional matrices.

The rank-revealing factorization provides an approach to low-rank approximation algorithms, which are directly derived from a truncated rank-revealing factorization. Because of the close relation between them, we do not distinguish *rank-revealing approximation* from *low-rank approximation* in terminology.

In general, a rank-revealing factorization is defined as follows.

Definition 15.3. For $\mathbf{A} \in \mathfrak{M}_{m,n}$, the factorization

$$\mathbf{A} = \mathbf{X}\mathbf{D}\mathbf{Y}' \quad (15.5)$$

is called a rank-revealing factorization of \mathbf{A} if $\mathbf{X} \in \mathfrak{M}_m$, $\mathbf{D} \in \mathfrak{D}_{m,n}$, $\mathbf{Y} \in \mathfrak{M}_n$, where \mathbf{D} is a diagonal matrix with nonnegative, non-increasing diagonal entries, and \mathbf{X} and \mathbf{Y} are well conditioned.

Let $d \leq \min(m, n)$, \mathbf{X}_d be the submatrix of \mathbf{X} consisting of the first d columns of \mathbf{X} , \mathbf{Y}_d be the submatrix of \mathbf{Y} consisting of the first d columns of \mathbf{Y} , and \mathbf{D}_d be the $d \times d$ leading submatrix of \mathbf{D} . Then the truncated factorization

$$\tilde{\mathbf{A}}_d = \mathbf{X}_d \mathbf{D}_d \mathbf{Y}'_d$$

is a d -rank approximation of \mathbf{A} provided \mathbf{X} and \mathbf{Y} both are “nearly” orthogonal. Here we use the word “nearly” orthogonal for a matrix with the conditional number near 1. Let $\mathbf{A} \in \mathfrak{M}_{m,n}$ have all singular values $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_k > 0$. When \mathbf{X} and \mathbf{Y} are nearly o.g., the diagonal entries of \mathbf{D} are close to σ_j , $1 \leq j \leq k$. Therefore, $\tilde{\mathbf{A}}_d$ approximates \mathbf{A}_d in (15.2).

There are various rank-revealing factorizations. For example,

- $\mathbf{A} = \mathbf{Q}\mathbf{R}\mathbf{P}'$ is called a rank-revealing QR factorization at index k if $\mathbf{Q} \in \mathfrak{D}_m$, $\mathbf{R} = \begin{pmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ 0 & \mathbf{R}_{22} \end{pmatrix}$ is upper triangular with $\mathbf{R}_{11} \in \mathfrak{M}_{k,k}$, $\mathbf{P} \in \mathfrak{M}_n$ is a permutation matrix, $\|\mathbf{R}_{22}\| = O(\sigma_{k+1})$, and $\sigma_k(\mathbf{R}_{11}) = O(\sigma_k)$.

- $\mathbf{A} = \mathbf{Q}'\mathbf{L}\mathbf{U}\mathbf{P}'$ is called a rank-revealing LU factorization at index k if \mathbf{P} and \mathbf{Q} are permutation matrices, $\mathbf{L} = \begin{pmatrix} \mathbf{L}_{11} & 0 \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{pmatrix}$ is lower triangular, $\mathbf{U} = \begin{pmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ 0 & \mathbf{U}_{22} \end{pmatrix}$ is upper triangular with $\mathbf{U}_{11}, \mathbf{L}_{11} \in \mathfrak{M}_k$, $\|\mathbf{L}_{22}\mathbf{U}_{22}\| = O(\sigma_{k+1})$, and $\sigma_k(\mathbf{L}_{11}\mathbf{U}_{11}) = O(\sigma_k)$.

A rank-revealing factorization at index k is also called k -rank revealing factorization. An algorithm realizing a rank-revealing factorization is called rank-revealing algorithm.

15.1.2 Fast Rank-revealing Algorithms

The role of the permutation matrices in the factorizations above is to search the principal k -subspace of $\text{Col}(\mathbf{A})$. Since the columns of the best k -rank approximation of \mathbf{A} spans the principal k -subspace, a general k -rank approximation of \mathbf{A} ought to span a subspace close to the principal one. A rank-revealing algorithm is fast if it can quickly find the approximatively principal k -subspace. The existence of fast rank-revealing algorithms is confirmed by many theorems. The following is one of them [1, 2].

Theorem 15.1. *Suppose $\mathbf{A} \in \mathfrak{M}_{m,n}$ and $k \leq \min(m, n)$. There is a matrix $\mathbf{P} \in \mathfrak{M}_{k,n}$ and a matrix $\mathbf{B} \in \mathfrak{M}_{m,k}$ whose columns constitute a subset of the columns of \mathbf{A} such that*

1. \mathbf{P} has a $k \times k$ submatrix, which is identity,
2. no entry of \mathbf{P} has an absolute value greater than 1,
3. $\|\mathbf{P}\| \leq \sqrt{k(n-k) + 1}$, and the least singular value of \mathbf{P} is ≥ 1 ,
4. $\mathbf{BP} = \mathbf{A}$, when $k = m$ or $k = n$, and $\|\mathbf{BP} - \mathbf{A}\| \leq \sqrt{k(n-k) + 1}\sigma_{k+1}$, when $k < \min(m, n)$, where σ_{k+1} is the $(k+1)$ th greatest singular value of \mathbf{A} .

It is also confirmed that reaching a nearly principal k column of \mathbf{A} only needs $Ckmn \log(n)$ operators.

Rank-revealing algorithms, such as rank-revealing QR algorithms, rank-revealing LU algorithms, and rank-revealing Cholesky decomposition algorithms, are well studied in the name of low-rank approximation in the numerical matrix theory. There are books on rank-revealing decompositions, e.g., [3, 4]. Many deterministic low-rank approximation algorithms already exist in literature. For example, Chan and Hansen [5], Gu and Eisenstat [6], Hong and Pan [7], Cheng, Gimbutas, Martinsson and Rokhlin [1], Berry, Pukatova, and Stewart [8], several papers of Goreinov, Tyrtyshnikov, and Zamarashkin [9–12], and many others. Fierro and Hansen developed two Matlab toolboxes, UTV Tools and UTV Expansion Pack, consisting of the rank-revealing algorithms, which are based on UTV factorization. Rank-

revealing UTV factorization is introduced in [13–16]. It is a modification of rank revealing SVD factorization. It can also be considered as a generalization of Schur decomposition. In UTV factorization the middle diagonal matrix Σ in SVD decomposition (15.1) is relaxed to a block-triangular matrix \mathbf{T} . If the middle matrix is upper triangular, then the decomposition is called URT factorization. Let the middle matrix in a URT decomposition be denoted by \mathbf{R} . For $m \leq n$, the factorization takes the form at index d as follows.

$$\mathbf{A} = \mathbf{U}_R \begin{pmatrix} \mathbf{R} \\ 0 \end{pmatrix} \mathbf{V}'_R = [\mathbf{U}_{Rd}, \mathbf{U}_{R0}, \mathbf{U}_{R\perp}] \begin{pmatrix} \mathbf{R}_d & \mathbf{F} \\ 0 & \mathbf{G} \\ 0 & 0 \end{pmatrix} \begin{bmatrix} \mathbf{V}_{Rd} \\ \mathbf{V}_{R0} \end{bmatrix}, \quad (15.6)$$

where $\mathbf{R}_d \in \mathfrak{M}_d$ is non-singular, \mathbf{G} is an $(n-d) \times (n-d)$ matrix. The URV decomposition is said to be d -rank revealing if

$$\sigma_d(R_d) = O(\sigma_d), \quad \|[\mathbf{F}', \mathbf{G}']\| = O(\sigma_{d+1}). \quad (15.7)$$

The truncated d -rank-revealing URV factorization

$$\mathbf{A}_{Rd} = \mathbf{U}_{Rd} \mathbf{R}_d \mathbf{V}'_{Rd} \quad (15.8)$$

is a d -rank-revealing approximation of \mathbf{A} . Let the SDV decomposition of \mathbf{R}_d be

$$\mathbf{R}_d = \mathbf{U}_d \mathbf{D}_d \mathbf{V}'_d.$$

Then

$$\mathbf{A}_{Rd} = (\mathbf{U}_{Rd} \mathbf{U}_d) \mathbf{D}_d (\mathbf{V}_{Rd} \mathbf{V}_d)' \stackrel{\text{def}}{=} \mathbf{U}_{Ad} \mathbf{D}_d \mathbf{V}'_{Ad}$$

is a d -rank-revealing SVD approximation of \mathbf{A} . Since both \mathbf{U}_d and \mathbf{V}_d are o.g., \mathbf{U}_{Rd} is a rotation of \mathbf{U}_{Ad} and \mathbf{V}_{Rd} is a rotation of \mathbf{V}_{Ad} .

If the middle upper triangular matrix \mathbf{R} in the URV decomposition is replaced by a lower triangular matrix \mathbf{L} , then the decomposition is called a ULV one. For $m \geq n$, at index d , the ULV decomposition takes the form

$$\mathbf{A} = \mathbf{U}_L \begin{pmatrix} \mathbf{L} \\ 0 \end{pmatrix} \mathbf{V}'_L = [\mathbf{U}_{Ld}, \mathbf{U}_{L0}, \mathbf{U}_{L\perp}] \begin{pmatrix} \mathbf{L}_d & 0 \\ \mathbf{H} & \mathbf{E} \\ 0 & 0 \end{pmatrix} \begin{bmatrix} \mathbf{V}_{Ld} \\ \mathbf{V}_{L0} \end{bmatrix}, \quad (15.9)$$

where $\mathbf{L}_d \in \mathfrak{M}_d$ is non-singular, \mathbf{E} is an $(n-d) \times (n-d)$ matrix. The ULV decomposition is said to be d -rank-revealing if

$$\sigma_{\min}(L_d) = O(\sigma_d), \quad \|[\mathbf{H}, \mathbf{E}]\| = O(\sigma_{d+1}). \quad (15.10)$$

The truncated d -rank-revealing ULV factorization

$$\mathbf{A}_{Ld} = \mathbf{U}_{Ld} \mathbf{L}_d \mathbf{V}'_{Ld} \quad (15.11)$$

is a d -rank-revealing approximation of \mathbf{A} . Let the SVD of \mathbf{L}_d be

$$\mathbf{L}_d = \tilde{\mathbf{U}}_d \mathbf{D}_d \tilde{\mathbf{V}}'_d.$$

Then

$$\mathbf{A}_{Ld} = (\mathbf{U}_{Ld} \tilde{\mathbf{U}}_d) \mathbf{D}_d (\mathbf{V}_{Ld} \tilde{\mathbf{V}}_d)' \stackrel{\text{def}}{=} \tilde{\mathbf{U}}_{Ad} \mathbf{D}_d \tilde{\mathbf{V}}'_{Ad}$$

is a d -rank-revealing SVD approximation of \mathbf{A} . When the gap between σ_d and σ_{d+1} is large, i.e., $\sigma_d \gg \sigma_{d+1}$, the d -rank-revealing SVD approximation in (15.2) is very close to the d -rank-revealing URV approximation in (15.8) and the d -rank-revealing ULV approximation in (15.11).

To estimate the approximation errors, we need the definition of the distance between two subspaces with the same dimension. Assume that S_1 and S_2 are two subspaces of \mathbb{R}^m with the same dimension. Let $\Theta(S_1, S_2)$ denote the angle between them. Then the distance between S_1 and S_2 can be defined by

$$d(S_1, S_2) = \sin(\Theta(S_1, S_2)).$$

For a matrix \mathbf{M} , we denote by $\sigma_d(\mathbf{M})$ the d th singular value of \mathbf{M} . The following estimates are given in Corollaries 2.3 and 2.5 of [13].

Theorem 15.2. *Let the d -rank-revealing UTV decomposition of \mathbf{A} be given in (15.8) and (15.11), and the SVD decomposition of \mathbf{A} be given in (15.1). Assume $\sigma_d(\mathbf{R}_d) > \|\mathbf{G}\|$. Then*

$$d(\text{Col}(\mathbf{U}_d), \text{Col}(\mathbf{U}_{Rd})) \leq \frac{\|\mathbf{F}\| \|\mathbf{G}\|}{\sigma_d^2(\mathbf{R}_d) - \|\mathbf{G}\|^2}$$

and

$$\frac{\|\mathbf{F}\|}{2\|\mathbf{R}\|} \leq d(\text{Col}(\mathbf{V}_d), \text{Col}(\mathbf{V}_{Rd})) \leq \frac{\sigma_d(\mathbf{R}_d) \|\mathbf{F}\|}{\sigma_d^2(\mathbf{R}_d) - \|\mathbf{G}\|^2}.$$

Similarly, assume $\sigma_{\min}(\mathbf{L}_d) > \|\mathbf{E}\|$. Then

$$\frac{\|\mathbf{H}\|}{2\|\mathbf{L}\|} \leq d(\text{Col}(\mathbf{U}_d), \text{Col}(\mathbf{U}_{Ld})) \leq \frac{\sigma_d(\mathbf{L}_d) \|\mathbf{H}\|}{\sigma_d^2(\mathbf{L}_d) - \|\mathbf{E}\|^2}$$

and

$$d(\text{Col}(\mathbf{V}_d), \text{Col}(\mathbf{V}_{Ld})) \leq \frac{\|\mathbf{H}\| \|\mathbf{E}\|}{\sigma_d^2(\mathbf{L}_d) - \|\mathbf{E}\|^2}.$$

We skip the proof of Theorem 15.2. Readers can find it in [13].

The following two UTL algorithms are given in [14]. The first low-rank algorithm L-ULV(L) is a natural “translation” of Stewart’s high-rank algorithm to the low-rank case so that it is “warm-started” with an initial ULV decomposition $\mathbf{A} = \mathbf{U}\mathbf{L}\mathbf{V}'$. Hence, the algorithm is named with the letter “L”. Usually, the initial matrix \mathbf{V} is chosen to equal the identity. When the algorithm terminates, the matrices \mathbf{H} and \mathbf{E} are of the forms $\mathbf{H} = [\hat{\mathbf{H}}, 0]'$ and $\mathbf{E} = [\hat{\mathbf{E}}, 0]'$, where $\hat{\mathbf{H}}$ and $\hat{\mathbf{E}}$ have $(n-d)$ rows. The following is the pseudo-code of the algorithm.

ALGORITHM L-ULV(L). Input: An $m \times n$ matrix \mathbf{A} and rank tolerance τ . Output: Numerical rank d , orthogonal matrices \mathbf{U} and \mathbf{V} , and $n \times n$ lower block-triangular matrix \mathbf{L} .

- Compute an initial ULV decomposition $\mathbf{A} = \mathbf{U}\mathbf{L}\mathbf{V}'$, let $i \leftarrow 1$.
- Compute estimates $\sigma_{\text{est}}^{(1)}$ and $\mathbf{u}_{\text{est}}^{(1)}$ of the largest singular value and corresponding left singular vector of the matrix \mathbf{L} .
- While $(\sigma_{\text{est}}^{(i)} > \tau \text{ and } i < n)$ do

Determine a sequence of Givens rotations $(\mathbf{P}_{n-i,n-i+1})_{i=1}^{n-1}$ such that $\mathbf{P}_{n-i,n-i+1} \cdots \mathbf{P}_{1,2} \mathbf{u}_{\text{est}}^{(i)} = \mathbf{e}_1$, where $\mathbf{e}_1 = [1, 0, \dots, 0]'$.

For $j = n - i : -1 : 2$,

$$\text{set } \hat{\mathbf{L}} \leftarrow \begin{pmatrix} \mathbf{I}_{i-1} & 0 \\ 0 & \mathbf{P}_{j-1,j} \end{pmatrix} \mathbf{L} \text{ and } \mathbf{U} \leftarrow \mathbf{U} \begin{pmatrix} \mathbf{I}_{i-1} & 0 \\ 0 & \mathbf{P}_{j-1,j} \end{pmatrix}.$$

Determine a Givens rotation $\mathbf{Q}_{j-1,j}$ such that $\hat{\mathbf{L}} \begin{pmatrix} \mathbf{I}_{i-1} & 0 \\ 0 & \mathbf{Q}_{j-1,j} \end{pmatrix}$ is lower triangular.

$$\text{Set } \mathbf{L} \leftarrow \hat{\mathbf{L}} \begin{pmatrix} \mathbf{I}_{i-1} & 0 \\ 0 & \mathbf{Q}_{j-1,j} \end{pmatrix} \text{ and } \mathbf{V} \leftarrow \mathbf{V} \begin{pmatrix} \mathbf{I}_{i-1} & 0 \\ 0 & \mathbf{Q}_{j-1,j} \end{pmatrix}.$$

- Deflate by setting $i \leftarrow i + 1$ and compute estimates $\sigma_{\text{est}}^{(i)}$ and $\mathbf{u}_{\text{est}}^{(i)}$ of the largest singular value and corresponding singular vector of $\mathbf{L}(i:n, i:n)$. End of ALGORITHM L-ULV(L).

The second “cold-started” low-rank algorithm avoids the initial factorization and starts directly from the matrix \mathbf{A} . Hence, the algorithm is named with the letter “A”. This algorithm is sometimes more efficient than the previous algorithm. When ALGORITHM L-ULV(A) terminates, both \mathbf{H} and \mathbf{E} are dense matrices with $m - d$ rows.

ALGORITHM L-ULV(A). Input: An $m \times n$ matrix \mathbf{A} and rank tolerance τ . Output: Numerical rank d , orthogonal matrices \mathbf{U} and \mathbf{V} and the $m \times n$ lower triangular matrix \mathbf{L} .

- Initialize $i \leftarrow 1$, $\mathbf{L} \leftarrow \mathbf{A}$, $\mathbf{U} \leftarrow \mathbf{I}_m$, and $\mathbf{V} \leftarrow \mathbf{I}_n$.
- Compute estimates $\sigma_{\text{est}}^{(1)}$ and $\mathbf{u}_{\text{est}}^{(1)}$ of the largest singular value and corresponding left singular vector of the matrix \mathbf{L} .
- While $(\sigma_{\text{est}}^{(i)} > \tau \text{ and } i < n)$, do

Determine a Householder matrix \mathbf{P}^i such that $\mathbf{P}^i \mathbf{u}_{\text{est}}^{(i)} = \mathbf{e}_1$.

$$\text{Set } \hat{\mathbf{L}} \leftarrow \begin{pmatrix} \mathbf{I}_{i-1} & 0 \\ 0 & \mathbf{P}^i \end{pmatrix} \mathbf{L} \text{ and } \mathbf{U} \leftarrow \mathbf{U} \begin{pmatrix} \mathbf{I}_{i-1} & 0 \\ 0 & \mathbf{P}^i \end{pmatrix}.$$

Determine a Householder matrix \mathbf{Q}^i that annihilates elements $i + 1$ through n of the i th row of $\hat{\mathbf{L}}$.

$$\text{Set } \mathbf{L} \leftarrow \hat{\mathbf{L}} \begin{pmatrix} \mathbf{I}_{i-1} & 0 \\ 0 & \mathbf{Q}^i \end{pmatrix} \text{ and } \mathbf{V} \leftarrow \mathbf{V} \begin{pmatrix} \mathbf{I}_{i-1} & 0 \\ 0 & \mathbf{Q}^i \end{pmatrix}.$$

- Deflate by setting $i \leftarrow i + 1$ and compute estimates $\sigma_{\text{est}}^{(i)}$ and $\mathbf{u}_{\text{est}}^{(i)}$ of the largest singular value and corresponding singular vector of $\mathbf{L}(i : n, i : n)$. End of ALGORITHM L-ULV(A).

The authors of [17] also developed UTV Expansion Pack as the supplement of the package UTV Tools. In UTV Expansion Pack, rank-revealing decompositions for positive semidefinite and indefinite matrices are included. Recall that the spectral decomposition of a symmetric $n \times n$ matrix \mathbf{A} with rank r is given as follows:

$$\mathbf{A} = \mathbf{V} \Lambda \mathbf{V}' = [\mathbf{V}_r, \mathbf{V}_\perp] \begin{pmatrix} \Lambda_r & 0 \\ 0 & 0 \end{pmatrix} [\mathbf{V}_r, \mathbf{V}_\perp]' = \mathbf{V}_r \Lambda_r \mathbf{V}_r', \quad (15.12)$$

where \mathbf{V} is o.g. and Λ_r is diagonal. We assume that the diagonal entries of Λ_r are descended arranged so that $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_r$. Then the best d -rank approximation of \mathbf{A} is

$$\mathbf{A}_d = \mathbf{V}_d \Lambda_d \mathbf{V}_d', \quad (15.13)$$

where \mathbf{V}_d consists of the first d columns of \mathbf{V} and $\Lambda_d = \text{diag}(\lambda_1, \dots, \lambda_d)$.

Symmetric rank-revealing VSV decompositions in [18, 19] provide the algorithms that take into account the symmetry of the matrix. Compared to the general UTV decompositions, the VSV decompositions lead to savings in computer time as well as advantages in the approximation properties of low-rank matrix approximations derived from the symmetric decompositions.

Assume that the positive demidefinite matrix \mathbf{A} has numerical rank d , i.e., $\lambda_d \gg \lambda_{d+1}$ and $\lambda_{d+1} \approx 0$. Then the rank-revealing VSV decomposition of \mathbf{A} takes the form

$$\mathbf{A} = \mathbf{V} \mathbf{S} \mathbf{V}', \quad \mathbf{S} = \begin{pmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{S}_{21} & \mathbf{S}_{22} \end{pmatrix}, \quad \mathbf{V} = [\mathbf{V}_1, \mathbf{V}_2], \quad (15.14)$$

where the symmetric matrix \mathbf{S} is partitioned such that it reveals the numerical rank of \mathbf{A} , i.e., the eigenvalues of the $d \times d$ leading submatrix \mathbf{S}_{11} approximate the first d eigenvalues of \mathbf{A} , while the norms \mathbf{S}_{12} and \mathbf{S}_{22} are both of the order λ_{d+1} . The matrix \mathbf{V} is orthogonal and partitioned such that the column spaces of the two blocks \mathbf{V}_1 and \mathbf{V}_2 approximate the subspaces spanned by the first d and the last $n - d$ right singular vectors of \mathbf{A} , respectively.

In the case of $d \ll n$, UTV Expansion Pack provides the low-rank VSV algorithms `1vsvid` and `1vsvsd`, which represent the middle matrix \mathbf{S} in (15.14) with the form

$$\mathbf{S} = \mathbf{T}' \boldsymbol{\Omega} \mathbf{T},$$

where \mathbf{T} is an upper or lower triangular matrix and $\boldsymbol{\Omega}$ is a diagonal matrix with ± 1 on the diagonal, such that the inertia of \mathbf{A} is preserved in the inertia of $\boldsymbol{\Omega}$. In `1vsvsd`, \mathbf{A} is positive semi-definite with the rank $\geq d$, then $\boldsymbol{\Omega}$ is

the identity matrix that can be omitted. The following is the pseudo-code of **lvsvid**.

Low-Rank VSV ALGORITHM. Input: An $n \times n$ symmetric data matrix \mathbf{A} and rank tolerance τ .

Output: Numerical rank d , orthogonal matrices \mathbf{V} , lower triangular matrix \mathbf{L} , and signature matrix $\boldsymbol{\Omega}$.

- Initialize $i \leftarrow 1$, $\mathbf{A} = \mathbf{L}'\boldsymbol{\Omega}\mathbf{L}$, $\mathbf{V} \leftarrow \mathbf{I}_n$.
- Compute estimates $\sigma_d = \|\mathbf{L}(d : n, d : n)' \boldsymbol{\Omega}(d : n, d : n) \mathbf{L}(d : n, d : n)\|$ and the corresponding right singular vector \mathbf{w}_d associated with σ_d .
- If ($\sigma_d < \tau$ then $k \leftarrow k - 1$), exit.
- Revealment: determine o.g. matrix \mathbf{P}_d such that $\mathbf{P}_d \mathbf{w}_d = \mathbf{e}_1$;
update $\mathbf{L}(d : n, d : n) \leftarrow \mathbf{L}(d : n, d : n) \mathbf{P}_d$ and $\mathbf{V}(:, d : n) \leftarrow \mathbf{V}(:, d : n) \mathbf{P}_d$;
make QL factorization of $\mathbf{L}(d : n, d : n)$: $\mathbf{L}(d : n, d : n) = \mathbf{Q}_d \mathbf{L}_d$.
update $\mathbf{L}(d : n, d : n) \leftarrow \mathbf{L}_d$ and $\boldsymbol{\Omega}(d : n, d : n) \leftarrow \mathbf{Q}'_d \boldsymbol{\Omega}(d : n, d : n) \mathbf{Q}_d$;
- Deflate by setting $d \leftarrow d + 1$.
- Go to step 2.

End of Low-Rank VSV ALGORITHM.

The pseudo-code of **lvsvid** is similar. The detailed introductions of these tool-boxes can be found in the manuals [15, 17]. The Matlab source code of UTV Tools and UTV Expansion Pack can be downloaded from the web sites of the authors of [15, 17]. Unfortunately, on the source code there are a few bugs that have to be fixed.

15.1.3 Nyström Approximation

Belabbas and Wolfe [20] proposed a fast low-rank approximation for covariance matrices. The method used in the approximation is called Nyström approximation, or shortly, Nyström extension [21–24]. The Nyström method is a technique for finding numerical approximations to eigenvectors of positive semi-definite matrices. To better understand the nature of the Nyström approximation, it is instructive to examine it from the standpoint of matrix completion. Let \mathbf{G} be an $N \times N$ positive semidefinite matrix, of which a partition is given by

$$\mathbf{G} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}' & \mathbf{C} \end{pmatrix} \quad (15.15)$$

with $\mathbf{A} \in \mathfrak{M}_n$, $\mathbf{B} \in \mathfrak{M}_{(N-n),n}$, and $\mathbf{C} \in \mathfrak{M}_{(N-n)}$. In the case of interest, $n \ll N$, so \mathbf{C} is huge. \mathbf{G} is psd, so is \mathbf{A} . Furthermore, the matrix \mathbf{A} most likely is positive definite so that \mathbf{A} has a spectral decomposition

$$\mathbf{A} = \mathbf{U} \boldsymbol{\Sigma} \mathbf{U}', \quad (15.16)$$

where Σ is invertible. The Nyström extension of \mathbf{U} is defined by

$$\bar{\mathbf{U}} = \begin{pmatrix} \mathbf{U} \\ \mathbf{B}'\mathbf{U}\Sigma^{-1} \end{pmatrix}, \quad (15.17)$$

whose columns approximate eigenvectors of \mathbf{G} . The matrix $\tilde{\mathbf{G}} = \bar{\mathbf{U}}\Sigma\bar{\mathbf{U}}'$ is an approximation of \mathbf{G} . It also takes the form

$$\begin{aligned} \tilde{\mathbf{G}} &= \bar{\mathbf{U}}\Sigma\bar{\mathbf{U}}' = \begin{bmatrix} \mathbf{U} \\ \mathbf{B}'\mathbf{U}\Sigma^{-1} \end{bmatrix} \Sigma [\mathbf{U}' \ \Sigma^{-1}\mathbf{U}'\mathbf{B}] \\ &= \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}' & \mathbf{B}'\mathbf{A}^{-1}\mathbf{B} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{A}' \\ \mathbf{B}' \end{bmatrix} \mathbf{A}^{-1} [\mathbf{A} \ \mathbf{B}]. \end{aligned}$$

Thus, the Nyström extension implicitly approximates \mathbf{C} by $\mathbf{B}'\mathbf{A}^{-1}\mathbf{B}$. The quality of the approximation can be quantified by the norm of the Schur complement $\|\mathbf{C} - \mathbf{B}'\mathbf{A}^{-1}\mathbf{B}\|$.

Note that the columns of $\bar{\mathbf{U}}$ in (15.17) are not orthogonal. To find an o.g. approximation of \mathbf{U} , we apply the following.

Theorem 15.3. *Let \mathbf{G} be a psd matrix having the partition (15.16), in which \mathbf{A} is pd. Define*

$$\mathbf{S} = \mathbf{A} + \mathbf{A}^{-1/2}\mathbf{B}\mathbf{B}'\mathbf{A}^{-1/2}$$

and decompose it as $\mathbf{S} = \mathbf{U}\Lambda\mathbf{U}'$. Let

$$\mathbf{V} = \begin{bmatrix} \mathbf{A} \\ \mathbf{B}' \end{bmatrix} \mathbf{A}^{-1/2}\mathbf{U}\Lambda^{-1/2}. \quad (15.18)$$

Then the columns of \mathbf{V} form an o.n. system, i.e., $\mathbf{V}'\mathbf{V} = \mathbf{I}$, and

$$\|\mathbf{G} - \mathbf{V}\Lambda\mathbf{V}'\| = \|\mathbf{C} - \mathbf{B}'\mathbf{A}^{-1}\mathbf{B}\|. \quad (15.19)$$

Proof. We have

$$\begin{aligned} \mathbf{V}'\mathbf{V} &= \left(\Lambda^{-1/2}\mathbf{U}'\mathbf{A}^{-1/2}[\mathbf{A} \ \mathbf{B}] \right) \left(\begin{bmatrix} \mathbf{A} \\ \mathbf{B}' \end{bmatrix} \mathbf{A}^{-1/2}\mathbf{U}\Lambda^{-1/2} \right) \\ &= (\Lambda^{-1/2}\mathbf{U}'\mathbf{A}^{-1/2})(\mathbf{A}^2 + \mathbf{B}\mathbf{B}')\mathbf{A}^{-1/2}\mathbf{U}\Lambda^{-1/2} \\ &= (\Lambda^{-1/2}\mathbf{U}')\mathbf{S}(\mathbf{U}\Lambda^{-1/2}) \\ &= \mathbf{I} \end{aligned}$$

and

$$\begin{aligned} \mathbf{V} \mathbf{A} \mathbf{V}' &= \begin{bmatrix} \mathbf{A} \\ \mathbf{B}' \end{bmatrix} \mathbf{A}^{-1/2} \mathbf{U} (\mathbf{U}' \mathbf{A}^{-1/2}) [\mathbf{A} \ \mathbf{B}] \\ &= \begin{bmatrix} \mathbf{A} \\ \mathbf{B}' \end{bmatrix} \mathbf{A}^{-1} [\mathbf{A} \ \mathbf{B}] \\ &= \tilde{\mathbf{G}}. \end{aligned}$$

The proof is completed. \square

More discussion about Nyström approximation can be found in [22, 24].

15.1.4 Greedy Low-rank Approximation

The analysis above tells us that it is possible to select some columns of \mathbf{G} to construct an approximation of \mathbf{G} . Without loss of generality, we assume that the first k columns of \mathbf{G} are the best selection for the construction of the approximation. Let $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_n]$ be the matrix such that $\mathbf{G} = \mathbf{A}\mathbf{A}'$. Write

$$\mathbf{G} = \sum_{j=1}^n \mathbf{a}_j \mathbf{a}'_j.$$

Then the best approximation of \mathbf{G} constructed by its first k columns is formulated so as to find a weight vector $\mathbf{w} = [w_1, \dots, w_k]'$ such that

$$\mathbf{w} = \arg \min_{\mathbf{w} \in \mathbb{R}^k} \left\| \mathbf{G} - \sum_{j=1}^k w_j \mathbf{a}_j \mathbf{a}'_j \right\|_{\text{F}}. \quad (15.20)$$

Let \mathbf{Q} be the Hadamard square of \mathbf{G} such that

$$\mathbf{Q} = \mathbf{G} \boxtimes \mathbf{G} = [\mathbf{Q}_{i,j}] = [\mathbf{G}_{i,j}^2], \quad (15.21)$$

which is partitioned into

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_k & \mathbf{Y} \\ \mathbf{Y}' & \mathbf{Z} \end{pmatrix}, \quad (15.22)$$

where $\mathbf{Q}_k \in \mathfrak{S}_k$ is the first major principal submatrix of \mathbf{Q} . Define the vector $\mathbf{r} \in \mathbb{R}^k$ by

$$\mathbf{r} = [\mathbf{Q}_k, \mathbf{Y}] \mathbf{1}. \quad (15.23)$$

Applying

$$\|\mathbf{G}\|_{\text{F}}^2 = \sum_{i,j=1}^n \mathbf{Q}_{i,j}, \quad \text{tr}(\mathbf{G} \mathbf{a}_i \mathbf{a}'_i) = \sum_{j=1}^n \mathbf{Q}_{i,j},$$

we have

$$\left\| \mathbf{G} - \sum_{j=1}^k w_j \mathbf{a}_i \mathbf{a}'_i \right\|_{\text{F}}^2 = \mathbf{w}' \mathbf{Q}_k \mathbf{w} - 2 \langle \mathbf{w}, \mathbf{r} \rangle + \sum_{i,j=1}^n \mathbf{Q}_{i,j},$$

which achieves the minimum at

$$\mathbf{w} = \mathbf{Q}_k^{-1} \mathbf{r}.$$

Let

$$\tilde{\mathbf{G}} = \sum_{j=1}^k w_j \mathbf{a}_i \mathbf{a}'_i \quad (15.24)$$

and

$$\tilde{\mathbf{Q}} = [\mathbf{Q}_k, \mathbf{Y}]' \mathbf{Q}_k^{-1} [\mathbf{Q}_k, \mathbf{Y}]. \quad (15.25)$$

We have

$$\|\mathbf{G} - \tilde{\mathbf{G}}\|_{\text{F}}^2 = \mathbf{1}' (\mathbf{Q} - \tilde{\mathbf{Q}}) \mathbf{1},$$

which provides the error estimate of the approximation. We state the obtained result in the following theorem.

Theorem 15.4. *Let \mathbf{G} be a positive semi-definite matrix and $\tilde{\mathbf{G}}$ be constructed as in (15.24). Let \mathbf{Q} , \mathbf{Q}_k , \mathbf{Y} , and \mathbf{Z} be defined as in (15.21) and (15.22). Let \mathbf{E} be the $(n-k) \times (n-k)$ matrix with all entries equal to 1. Then*

$$\|\mathbf{G} - \tilde{\mathbf{G}}\|_{\text{F}}^2 = \text{tr}((\mathbf{Z} - \mathbf{Y}' \mathbf{Q}_k \mathbf{Y}) \mathbf{E}). \quad (15.26)$$

By Theorem 15.4, the greedy low-rank matrix approximation is designed as follows.

Assume that the positive semidefinite $n \times n$ matrix \mathbf{G} and the dimension k are given ($k < n$).

Step 1. Select the greedy columns. Write $\mathbf{G} = [g_1 \cdots g_n]$. Compute the norm vector $\mathbf{T} = [\|g_i\|^2]_{i=1}^n$ and find the index vector $\mathbf{J} = [j_1, \dots, j_k]$, where the columns have k largest norms ($k < n$).

Step 2. Construct \mathbf{Q} and \mathbf{Q}_k . Set $\mathbf{Q}_{i,j} = Q_{i,j}^2$, $1 \leq i, j \leq n$ and $\mathbf{Q}_k = [Q_{j,l}]_{j,l \in J}$.

Step 3. Compute vector \mathbf{r} . Set $r_l = \sum_{j=1}^n Q_{l,j}$, $l \in J$.

Step 4. Compute weight vector \mathbf{w} . Set $\mathbf{w} = \mathbf{Q}_k^{-1} \mathbf{r}$.

Step 5. Compute \mathbf{F} so that \mathbf{FF}' approximate \mathbf{G} . Set $\mathbf{D} = \text{diag}(\mathbf{w})$, $\mathbf{G}_k = [g_{j_1} \cdots g_{j_k}]$, and $\mathbf{F} = \mathbf{G}_k \mathbf{D}$. Compute the SVD of $\mathbf{F} = \mathbf{U} \Sigma \mathbf{V}'$, where $\mathbf{U} \in \mathfrak{O}_{n,k}$.

Step 6. Construct the approximation $\tilde{\mathbf{G}}$. Set $\tilde{\mathbf{G}} = \mathbf{U} \mathbf{U}' \mathbf{G} \mathbf{U} \mathbf{U}'$.

15.2 Randomized Algorithm for Matrix Approximation

15.2.1 Randomized Low-rank Approximation

The low-rank approximation algorithms introduced in the last section are deterministic. There exist also randomized algorithms for matrix approximation. Randomized algorithms usually are faster than the deterministic ones when they are applied to solve the same problems. As a trade-off, the randomized algorithms are successful in a probability.

We remark that JL-embedding algorithms introduced in Chapter 7 cannot be directly applied to low-rank approximation. Firstly, by Theorem 7.1 and Theorem 7.2, the dimension d of the reduced data has to obey the inequality (7.6), it cannot be as small as desired. For instance, if $n = 10000$ and $\varepsilon = 1/2$, then, by the estimate in Theorem 7.1 (also in Theorem 7.2), the dimension d has to be restricted to $d > 331$, which is still quite large in many tasks. Secondly, JL-embedding algorithms do not have the functionality of locating feature vectors, and therefore are not reliable in data-geometry preservation.

Martinsson, Rokhlin, and Tygert [2, 25] proposed a randomized algorithm for approximating SVD decomposition. In their algorithm, a random projection maps the original high-dimensional matrix onto a low-dimensional one, so that the SVD of the transformed matrix approximates the SVD of the original one. More precisely, let \mathbf{X} be a given $m \times n$ matrix, where both m and n are very large. Let $\mathcal{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_k\}$ be an o.n. basis of a random k -dimensional subspace $S_k \subset \mathbb{R}^m$, where $k \ll \min(m, n)$. Let $\mathbf{B} = [\mathbf{b}_1 \dots \mathbf{b}_k]$. The projections of the column vectors of \mathbf{X} onto S_k have the \mathcal{B} -coordinates $\mathbf{X}_B = \mathbf{B}'\mathbf{X}$. It can be proved that $\mathbf{B}\mathbf{X}_B$ is a k -rank revealing approximation of \mathbf{X} with a certain probability. Then the SVD of $\mathbf{B}\mathbf{X}_B$ approximates the SVD of \mathbf{X} . Note that matrix \mathbf{X}_B has the dimension $k \times n$, where k is much smaller than both m and n . Hence, the cost for computing the SVD of \mathbf{X}_B is much lower than the cost for the SVD of \mathbf{X} . The project technique used above can also be replaced by interpolation.

The authors of [25] proposed two randomized algorithms for low-rank approximation of matrices. One adopts interpolative decompositions, the other uses projections. We shall introduce them in the next two subsections. In the following, we assume that the matrix \mathbf{A} has dimension $m \times n$ and rank r with $r \ll \min(m, n)$, and both m and n are large. We denote by σ_j the j th singular value of \mathbf{A} .

15.2.2 Randomized Interpolative Algorithm

The randomized interpolative low-rank approximation produces an interpolative $m \times k$ matrix \mathbf{P} and a $k \times n$ matrix \mathbf{B} , which are similar to \mathbf{P} and \mathbf{B} in Theorem 15.1. The existing algorithms for computing \mathbf{B} and \mathbf{P} in Theorem 15.1 are computationally expensive. In [25], the authors suggested to use the rank-revealing algorithms in [1, 6, 26] to produce \mathbf{B} and \mathbf{P} , which satisfy somewhat weaker conditions than those in Theorem 15.1. The algorithms compute \mathbf{B} and \mathbf{P} such that

- some subset of the columns of \mathbf{P} makes up the $k \times k$ identity matrix,
- no entry of \mathbf{P} has an absolute value greater than 2,
- $\|\mathbf{P}\| \leq c_{n,k}$,
- the least singular value of \mathbf{P} is at least 1,
- $\mathbf{B}\mathbf{P} = \mathbf{A}$ when $k = m$ or $k = n$, and
- $\|\mathbf{B}\mathbf{P} - \mathbf{A}\| \leq c_{n,k}\sigma_{k+1}$ when $k < \min(m, n)$,

where

$$c_{n,k} = \sqrt{4k(n-k) + 1}. \quad (15.27)$$

The algorithms of this type are based upon the Cramer's rule to obtain the minimal norm (or at least nearly minimal-norm) solutions of linear systems. The details refer to [1, 6, 26]. Comparing the conditions above with those in Theorem 15.1, we can find two differences that (1) the upper bound $\|\mathbf{P}\| \leq \sqrt{k(n-k) + 1}$ now is relaxed to $\|\mathbf{P}\| \leq c_{n,k}$ and (2) $\max(|\mathbf{P}_{i,j}|) \leq 1$ is relaxed to $\max(|\mathbf{P}_{i,j}|) \leq 2$. Besides, for any positive real number ε , the algorithms can identify the least k such that $\|\mathbf{B}\mathbf{P} - \mathbf{A}\| \approx \varepsilon$. An algorithm of this type is computationally economic: there exists a positive real number C so that the algorithm computes both \mathbf{B} and \mathbf{P} using at most $Ckmn \log(n)$ floating-point operations.

Let \mathbf{A} be a given high-dimensional matrix. According to the discussion above, a randomized algorithm of interpolative low-rank approximation uses a random projection to produce a low-dimensional projection of \mathbf{A} , then applies a rank-revealing algorithm to find the submatrix \mathbf{B} of \mathbf{A} and the corresponding interpolative matrix \mathbf{P} , such that $\mathbf{B}\mathbf{P}$ is a low-rank approximation of \mathbf{A} . The randomized interpolative algorithm consists of the following steps.

Assume that an $m \times n$ matrix \mathbf{A} is given. Both m and n are large. We set $l < \min(m, n)$ and $k < l$.

Step 1. Create a random projection of \mathbf{A} . Create an $l \times m$ random matrix \mathbf{G} whose entries are i.i.d. Gaussian random variables of zero mean and unit variance, and compute the $l \times n$ product matrix $\mathbf{R} = \mathbf{G}\mathbf{A}$.

Step 2. Find the interpolative matrix \mathbf{P} . Using a rank-revealing algorithm to create a real $l \times k$ matrix \mathbf{S} and a real $k \times n$ matrix \mathbf{P} , where the columns of \mathbf{S} constitute a subset of the columns of \mathbf{R} and \mathbf{P} satisfies $\|\mathbf{P}\| \leq c_{n,k}$, such that

$$\|\mathbf{S}\mathbf{P} - \mathbf{R}\| \leq c_{n,k}\rho_{k+1}, \quad (15.28)$$

where ρ_{k+1} is the $(k+1)$ st greatest singular value of \mathbf{R} .

Step 3. Find the submatrix \mathbf{B} of \mathbf{A} . Due to Step 2, the columns of \mathbf{S} constitute a subset of the columns of \mathbf{R} . Let $J = \{i_1, \dots, i_k\}$ be the index set such that the j th column of \mathbf{S} is the i_j th column of \mathbf{R} . Create an $m \times k$ submatrix \mathbf{B} of \mathbf{A} , so that the j th column of \mathbf{B} is the i_j th column of \mathbf{A} . The matrix \mathbf{B} can be computed by $\mathbf{B} = \mathbf{AP}'$.

Then $\tilde{\mathbf{A}} \stackrel{\text{def}}{=} \mathbf{BP} = \mathbf{AP}'\mathbf{P}$ is a k -low-rank approximation of \mathbf{A} .

We give the error estimate of the approximation realized by the algorithm above. Let β and γ be two positive real numbers such that $\gamma > 1$ and define

$$\tilde{p}(l, k, \beta) = \frac{1}{\sqrt{2\pi(l-k+1)}} \left(\frac{e}{(l-k+1)\beta} \right)^{(l-k+1)}, \quad (15.29)$$

$$p(x, \gamma) = \frac{1}{4(\gamma-1)\sqrt{\pi x \gamma}} \left(\frac{2\gamma}{e^{\gamma-1}} \right)^x. \quad (15.30)$$

We have the following error estimate of the approximation given by the algorithm above.

Theorem 15.5. *Let \mathbf{A} be an $m \times n$ matrix, $k \leq l < \min(n, m)$. Let \mathbf{P} and \mathbf{B} be the matrices that are produced by the randomized algorithm of interpolative low-rank approximation above. Then*

$$\|\mathbf{A} - \mathbf{BP}\| \leq \left(\sqrt{2lm\beta\gamma + 1} (c_{n,k} + 1) + \sqrt{2lm\beta\gamma} c_{n,k} \right) \sigma_{k+1} \quad (15.31)$$

with probability at least χ , where $c_{n,k}$ is in (15.27), σ_{k+1} is the $(k+1)$ th singular value of \mathbf{A} , and

$$\chi = 1 - \tilde{p}(l, k, \beta) - 2p(m, \gamma). \quad (15.32)$$

When the singular values decay fast. We can get the following more accurate estimate.

Theorem 15.6. *Let \mathbf{A} , \mathbf{P} , and \mathbf{B} be the matrices as in Theorem 15.5. Assume that $j > 0$ satisfies $k+j < \min(m, n)$. Write $m_1 = \max(m-k-j, l)$, $m_2 = \max(j, l)$, and $m_3 = \max(j+k, l)$. Then*

$$\|\mathbf{A} - \mathbf{BP}\| \leq \xi \sigma_{k+1} + \eta \sigma_{k+j+1} \quad (15.33)$$

with probability at least κ , where

$$\kappa = 1 - \tilde{p}(l, k, \beta) - 2p(m_1, \gamma) - p(m_2, \gamma) - p(m_3, \gamma) \quad (15.34)$$

and

$$\xi = \sqrt{2l\beta\gamma m_2 + 1} (c_{n,k} + 1) + \sqrt{2l\beta\gamma m_3} c_{n,k}, \quad (15.35)$$

$$\eta = \sqrt{2l\beta\gamma m_1 + 1} (c_{n,k} + 1) + \sqrt{2l\beta\gamma m_1} c_{n,k}. \quad (15.36)$$

Moreover, if $\eta\sigma_{k+j+1}$ is less than $\xi\sigma_{k+1}$, then

$$\|\mathbf{A} - \mathbf{B}\mathbf{P}\| \leq 2\xi\sigma_{k+1} \quad (15.37)$$

with probability at least κ .

In practice, we usually choose $l = k + 20$. In this case we have the following.

Corollary 15.1. *In Theorem 15.5, if we choose $l = k + 20$, $\beta = 9/16$, and $\gamma = 5$, then*

$$\|\mathbf{B}\mathbf{P} - \mathbf{A}\| < 10\sqrt{k(k+20)mn\sigma_{k+1}}. \quad (15.38)$$

with probability not less than $1 - 10^{-17}$.

15.2.3 Randomized SVD Algorithm

The randomized algorithm of SVD low-rank approximation uses a random projection to project \mathbf{A} on a random subspace of $\text{Row}(\mathbf{A})$, then applies a SVD algorithm to yield the SVD of the projection of \mathbf{A} . Since the the projected matrix has a much lower dimension than \mathbf{A} , the algorithm performs fast. The randomized algorithm consists of the following steps.

Assume that an $m \times n$ matrix \mathbf{A} is given. Both m and n are large. We set $l < \min(m, n)$ and $k < l$.

Step 1. Create a random projection of \mathbf{A} . Create an $l \times m$ random matrix \mathbf{G} whose entries are i.i.d. Gaussian random variables of zero mean and unit variance, and compute the $l \times n$ product matrix $\mathbf{R} = \mathbf{G}\mathbf{A}$.

Step 2. Find approximative principal subspace \mathbf{L} . Using an SVD algorithm to find a real $n \times k$ matrix \mathbf{Q} and a $k \times l$ matrix \mathbf{S} , where the columns of \mathbf{Q} are orthonormal and

$$\|\mathbf{QS} - \mathbf{R}'\| \leq \rho_{k+1}, \quad (15.39)$$

where ρ_{k+1} is the $(k+1)$ st greatest singular value of R . Let $\mathbf{L} = \text{Col}(\mathbf{Q})$, which approximates the the principal k -subspace of \mathbf{A} . The columns of \mathbf{Q} form an o.n. basis of \mathbf{L} .

Step 3. Find o.g. projection \mathbf{T} from \mathbf{A} to \mathbf{L} . \mathbf{T} is given by $\mathbf{T} = \mathbf{AQ}$.

Step 4. Find SVD of \mathbf{T} . Let $\mathbf{T} = \mathbf{U}\Sigma\mathbf{W}'$ be the SVD of \mathbf{T} .

Step 5. Find low-rank approximation of SVD of \mathbf{A} . Let $\mathbf{V} = \mathbf{Q}\mathbf{W}$. Then the triple $[\mathbf{U}, \Sigma, \mathbf{V}]$ is a low-rank SVD approximation of \mathbf{A} : $\mathbf{U}\Sigma\mathbf{V}' \approx \mathbf{A}$.

We have the following error estimate of the approximation given by the algorithm above.

Theorem 15.7. *Let \mathbf{A} be an $m \times n$ matrix, $k \leq l < \min(n, m)$. Let the triple $[\mathbf{U}, \Sigma, \mathbf{V}]$ is produced by the randomized algorithm of low-rank SVD*

approximation above. Then

$$\|\mathbf{A} - \mathbf{U}\Sigma\mathbf{V}'\| \leq 2 \left(\sqrt{2lm\beta\gamma + 1} + \sqrt{2lm\beta\gamma} \right) \sigma_{k+1} \quad (15.40)$$

with probability at least χ .

When the singular values of \mathbf{A} decay fast, a better error estimate is given in the following theorem.

Theorem 15.8. *Let \mathbf{A} , \mathbf{U} , Σ , and \mathbf{V} be the matrices in Theorem 15.7, and let $j > 0$ satisfy $k + j < \min(m, n)$. Write $m_1 = \max(m - k - j, l)$, $m_2 = \max(j, l)$, and $m_3 = \max(j + k, l)$. Then*

$$\|\mathbf{A} - \mathbf{U}\Sigma\mathbf{V}'\| \leq \xi\sigma_{k+1} + \eta\sigma_{k+j+1}, \quad (15.41)$$

with probability at least κ given in (15.34), and

$$\xi = 2 \left(\sqrt{2l\beta\gamma m_2 + 1} + \sqrt{2l\beta\gamma m_3} \right), \quad (15.42)$$

$$\eta = 2 \left(\sqrt{2l\beta\gamma m_1 + 1} + \sqrt{2l\beta\gamma m_1} \right). \quad (15.43)$$

Moreover, if $\eta\sigma_{k+j+1}$ is less than $\xi\sigma_{k+1}$, then

$$\|\mathbf{A} - \mathbf{U}\Sigma\mathbf{V}'\| \leq 2\xi\sigma_{k+1} \quad (15.44)$$

with probability at least κ .

Similarly, we have the following.

Corollary 15.2. *In Theorem 15.7, if we choose $l = k + 20$, $\beta = 9/16$, and $\gamma = 5$, then*

$$\|\mathbf{A} - \mathbf{U}\Sigma\mathbf{V}'\| < 10\sqrt{k(k + 20)m}\sigma_{k+1}. \quad (15.45)$$

with probability not less than $1 - 10^{-17}$.

The proofs of Theorems 15.5–15.8 are rather technical. We shall put the proofs of Theorems 15.7 and 15.8 in the last section of the chapter. Since the proofs of Theorems 15.5 and 15.6 are similar, they will be omitted.

15.2.4 Randomized Greedy Algorithm

The deterministic greedy low-rank approximation algorithm introduced in the last section can be modified to a randomized one. The idea is very simple: Change the greedy selection of columns in \mathbf{G} to a random selection. Assume that \mathbf{G} is an $n \times n$ positive semi-definite matrix. The algorithm randomly selects an index set $J = \{i_1, \dots, i_k\}$ for the approximation. Since

\mathbf{G} is positive semi-definite, it induces that the probability distribution on the set of all J with $|J| = k$ is

$$p(J) = \frac{\det(\mathbf{G}_J)}{\sum_{|L|=k} \det(\mathbf{G}_L)}, \quad (15.46)$$

where $\det(\mathbf{G}_L)$ is the determinant of the submatrix \mathbf{G}_L whose rows and columns are chosen from the index set L .

The randomized greedy algorithm for low-rank matrix approximation consists of the same steps as the deterministic one except that the index now is randomly selected. The algorithm is well behaved in the sense that if \mathbf{G} is of rank k , then $\tilde{\mathbf{G}} = \mathbf{G}$. To prove this, we mention that $\det(\mathbf{G}_J) \neq 0$ implies $\text{rank}(\mathbf{G}_J) = k$, and then $\|\mathbf{C} - \mathbf{B}\mathbf{A}^{-1}\mathbf{B}'\|_F = 0$ in (15.49) which gives $\tilde{\mathbf{G}} = \mathbf{G}$.

For the general case whereupon $\text{rank}(\mathbf{G}) > k$, we have the following error bound in expectation.

Theorem 15.9. *Let \mathbf{G} be a real, $n \times n$, positive semidefinite matrix with eigenvalues $\lambda_1 \geq \dots \geq \lambda_n$. Let $\tilde{\mathbf{G}}$ be the Nyström approximation to \mathbf{G} corresponding to J , with $p(J)$ as given in (15.46). Then*

$$\mathbb{E}(\|\mathbf{G} - \tilde{\mathbf{G}}\|_F) \leq (k+1) \sum_{l=k+1}^n \lambda_l,$$

with the probability $p(J)$.

The proof of the theorem is quite similar to that for Theorem 15.4. We skip the proof and the readers may refer to [27].

15.3 Fast Anisotropic Transformation DR Algorithms

In this section, we introduce fast anisotropic transformation (FAT) DR algorithms, which is based on low-rank approximation technique.

15.3.1 Fast Anisotropic Transformation

Proposed by Chui and Wang in [28, 29], FAT provides a framework for developing fast DR algorithms. Recall that in each of nonlinear DR methods introduced previously, the kernel is an $n \times n$ matrix, where n is the total number of points in the observed data, which can be very large in many applications. For example, even for a small HSI cube with $560 \times 480 = 268800$ raster cells, when the spectral curves are selected as the objective vectors, the number of entries of a nonlinear DR kernel is almost 7×10^{10} . Hence, when nonlinear DR methods are applied to a larger data set, they usually encounter difficulties

in the three aspects: memory usage, computational complexity, and computational instability. Although some nonlinear DR kernels are sparse matrices, which enable us partially to overcome the difficulties in memory usage and computational complexity, yet it is not clear if the instability issue can be settled. The aim of fast anisotropic transforms is to avoid spectral decomposition of large-dimensional DR kernels to dramatically reduce memory usage and computational complexity, as well as to increase numerical stability.

FAT is based on the observation of the particular structures of DR kernels. Over all of the discussed nonlinear DR kernels, we can computationally classify them into two types. In the first type are Isomaps, MVU, and Dmaps, which find the DR data from the leading eigenvectors of their kernels. In the second type are LLE, LTSA, Leigs, and HLLE, which find the DR data from their bottom eigenvectors. To find the DR data, the methods of the first type establish maximization models while the methods of the second type adopt minimization models. A modeling duality view of nonlinear DR methods is discussed in [30]. Using the duality of maximization and minimization, we can convert the minimization problems in LLE, LTSA, Leigs, and HLLE to maximization ones. Therefore, in the following discussion, we assume that the DR kernels of the second type have been converted so that the DR data in these methods are also obtained from the several leading eigenvectors of the converted kernels. A slight difference still exists for such kernels. For example, in Isomaps the DR data are obtained from the d -leading eigenvectors, but in Dmaps from the 2nd to the $(d+1)$ th ones. Surely, this difference does not impact our FAT framework very much.

The main idea of FAT is similar to diffusion maps. Assume that a data set $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^D$ (nearly) resides on a d -dimensional manifold $M \subset \mathbb{R}^D$, whose geometry is described by a DR kernel \mathbf{K} . Since \mathbf{K} is positive semi-definite, we can define the kernel distance between \mathbf{x}_i and \mathbf{x}_j by

$$d_{\mathbf{K}}^2(\mathbf{x}_i, \mathbf{x}_j) = k_{ii} + k_{jj} - 2k_{ij}. \quad (15.47)$$

Assume the rank of \mathbf{K} is m . In the ideal case $m = d$. Otherwise $d < m$. Let a Cholesky decomposition of \mathbf{K} be

$$\mathbf{K} = \mathbf{P}' \mathbf{P},$$

where $\mathbf{P} = [\mathbf{p}_1, \dots, \mathbf{p}_n] \in \mathfrak{M}_{m,n}$ corresponds to the data $\mathcal{P} = \{\mathbf{p}_1, \dots, \mathbf{p}_n\} \subset \mathbb{R}^m$. Then

$$d_2(\mathbf{p}_i, \mathbf{p}_j) = d_K(\mathbf{x}_i, \mathbf{x}_j).$$

Let $\mathbf{h} = [h^1, \dots, h^d]$ be the manifold coordinate mapping. Then the DR data are represented by $\mathcal{Y} = h(\mathcal{X})$. If h is an isometric, \mathcal{Y} exactly preserves the kernel distance on \mathcal{X} ,

$$d_2(\mathbf{y}_i, \mathbf{y}_j) = d_K(\mathbf{x}_i, \mathbf{x}_j),$$

which yields

$$d_2(\mathbf{y}_i, \mathbf{y}_j) = d_2(\mathbf{p}_i, \mathbf{p}_j).$$

In general, the equation above only holds in the approximative sense. Let the SVD of \mathbf{P} be

$$\mathbf{P} = \sum_{i=1}^m \sigma_i \mathbf{u}_i \mathbf{v}'_i. \quad (15.48)$$

Write $\mathbf{V} = [\mathbf{v}_1 \cdots \mathbf{v}_m]$. Then \mathbf{Y} can be chosen as $\mathbf{Y} = [\mathbf{v}_1 \cdots \mathbf{v}_d]'$. Note that the column space $S = \text{Col}(\mathbf{V})$ (which is equal to $\text{Row}(\mathbf{P})$) is an m -dimensional subspace of \mathbb{R}^n , which “wraps” coordinate functions h^1, \dots, h^d .

Definition 15.4. Let $\mathcal{X} \subset \mathbb{R}^D$ be a given data set, which resides on a d -dimensional manifold $M \subset \mathbb{R}^D$. Let $h = [h^1, \dots, h^d]'$ be a coordinate mapping on M . An m -dimensional subspace $S \subset \mathbb{R}^n$ is called a wrapped-coordinate subspace if $\mathbf{h}^i = h^i(\mathcal{X}) \in S, 1 \leq i \leq d$. A function $f = [f^1, \dots, f^m]'$ is called an anisotropic transformation if the vector set $\{f^i = f^i(\mathcal{X})\}_{1 \leq i \leq m}$ spans S . Let $\mathbf{p}_j = f(\mathbf{x}_j), 1 \leq j \leq n$. Then \mathbf{p}_j is called a wrapped-coordinate representation of \mathbf{x}_j and $P = [\mathbf{p}_1 \cdots \mathbf{p}_n]$ is called a wrapped DR data matrix.

An FAT algorithm first constructs the wrapped DR data matrix \mathbf{P} , and then finds the DR data \mathbf{Y} from \mathbf{P} using a linear method, say, PCA. Since the SVD decomposition in an FAT algorithm is only applied to the low-dimensional matrix \mathbf{P} , the algorithm is fast. Note that the wrapped DR data matrix \mathbf{P} is not unique. There are many different “wraps” for the same DR data. The variety of \mathbf{P} offers the freedom for developing various FAT DR algorithms.

15.3.2 Greedy Anisotropic Transformation

The first type of FAT algorithms is based on the greedy low-rank approximation. Hence, we call them greedy anisotropic transformation (GAT) algorithms, which are described as follows.

Let $\mathbf{P} = [\mathbf{p}_1 \cdots \mathbf{p}_n] \in \mathfrak{M}_{m,n}$ be a wrapped DR data matrix. The relation $\mathbf{K} = \mathbf{P}'\mathbf{P}$ yields $\text{Row}(\mathbf{P}) = \text{Col}(\mathbf{K})$. Therefore, there are m columns in \mathbf{K} spanning $\text{Row}(\mathbf{P})$. GAT algorithm selects the m greedy columns of \mathbf{K} to (approximately) span $\text{Row}(\mathbf{P})$. Write $\mathbf{K} = [\mathbf{k}_1 \cdots \mathbf{k}_n]$. Without losing generality, we assume that $\mathbf{k}_1, \dots, \mathbf{k}_m$ are m greedy columns of \mathbf{K} . Denote by S the subspace spanned by these vectors. Let $\{\mathbf{b}_1, \dots, \mathbf{b}_m\}$ be an o.n. basis of S . Then \mathbf{p}_j can be chosen as the o.g. projection of \mathbf{x}_j on S , represented in \mathcal{B} -coordinates. The SVD of \mathbf{P} yields the DR data matrix \mathbf{Y} .

Remark 15.1. If the m columns of \mathbf{K} are chosen randomly, then we obtain a randomized greedy anisotropic transformation algorithm, which will be discussed in the next subsection.

The pseudo-code of the GAT algorithm is the following.

Input: the positive semi-definite $n \times n$ kernel \mathbf{K} , the dimension of wrapped-coordinate space m , and the dimension of the DR data d .

Output: the DR data matrix \mathbf{Y} and the corresponding leading eigenvalue set ev .

Step 1. Find greedy columns. Compute the norm of each column vector of \mathbf{K} and construct the $n \times m$ submatrix \mathbf{K}_m that consists of the m columns that have m largest norms.

Step 2. Make wrapped DR data matrix. Make the QR factorization of \mathbf{K}_m to obtain a matrix $\mathbf{Q} \in \mathfrak{O}_{n,m}$ whose columns form an o.n. basis of $\text{Col}(\mathbf{K}_m)$. Then the wrapped DR data matrix is $\mathbf{A} = (\mathbf{K}\mathbf{Q})'$.

Step 3. Find DR data. Let the SVD decomposition of \mathbf{A} be

$$\mathbf{A} = \sum_{i=0}^m \sigma_i \mathbf{v}^i (\mathbf{u}^i)',$$

where $\sigma_0 > \sigma_1 \geqslant \dots \geqslant \sigma_m > 0$. If \mathbf{K} is an Isomaps-type kernel, then $\mathbf{Y}' = [\mathbf{v}^0 \ \dots \ \mathbf{v}^{d-1}]$. Otherwise, if \mathbf{K} is a Dmaps-type kernel, we have $\sigma_0 = 1$. Setting $\mathbf{Y}' = [\mathbf{v}^1 \ \dots \ \mathbf{v}^d]$, and then we entry-wise divide each \mathbf{v}_j by \mathbf{v}_0 .

15.3.3 Randomized Anisotropic Transformation

Randomized anisotropic transformation (RAT) algorithms construct the approximative wrapped-coordinate subspace S by random projection or random interpolation. The randomized interpolative anisotropic transformation (I-RAT) algorithm randomly selects m columns of \mathbf{K} to span the subspace S , while the randomized project anisotropic transformation (P-RAT) algorithm randomly selects an m -subspace of $\text{Col}(\mathbf{K})$ to be the subspace S . Once the subspace S is constructed, the remaining steps in the algorithms are the same as in GAT algorithm. The pseudo-codes of the algorithms are in the following.

Input: the positive semi-definite kernel \mathbf{K} , the dimension of the wrapped-coordinate space m , and the dimension of the DR data d .

Output: the DR data matrix \mathbf{Y} and the corresponding eigenvalues ev .

Step 1. Find data-wrapping subspace. If the interpolative method is applied, randomly select m columns of \mathbf{K} to make the $n \times m$ submatrix \mathbf{K}_m . If the project method is selected, then create an $n \times m$ random matrix \mathbf{R} and set $\mathbf{K}_m = \mathbf{K}\mathbf{R}$.

Step 2. Make DR data-wrap matrix. Make the QR factorization of \mathbf{K}_m to obtain a matrix $\mathbf{Q} \in \mathfrak{O}_{n,m}$ whose columns form an o.n. basis of $\text{Col}(\mathbf{K}_m)$. Then make the the wrapped DR data matrix $\mathbf{A} = (\mathbf{K}\mathbf{Q})'$.

Step 3. Find DR data. Let the SVD decomposition of \mathbf{A} be

$$\mathbf{A} = \sum_{i=0}^m \sigma_i \mathbf{v}^i (\mathbf{u}^i)',$$

where $\sigma_0 > \sigma_1 \geqslant \cdots \geqslant \sigma_m > 0$. If \mathbf{K} is an Isomaps-type kernel, then $\mathbf{Y}' = [\mathbf{v}^0 \ \cdots \ \mathbf{v}^{d-1}]$. Otherwise, if \mathbf{K} is a Dmaps-type kernel, set $\mathbf{Y}' = [\mathbf{v}^1 \ \cdots \ \mathbf{v}^d]$, and then divide each \mathbf{v}_j entry-wise by \mathbf{v}_0 .

Remark 15.2. The steps 2 and 3 in RAT algorithm are the same as in GAT algorithm. The algorithms can be also used for linear DR of a data set \mathcal{X} . In the case that the input matrix is the $D \times n$ data matrix \mathbf{X} , we use $\mathbf{K} = \mathbf{X}'$ in the first step of RAT.

The computational cost of an FAT algorithm (GAT or RAT) is economic. Let the cost be measured by the operational times. Assume that \mathbf{G} is a general $n \times c$ matrix, which is not necessary to be positive semi-definite. Let C_G denote the cost of the multiplication of \mathbf{G} to a c -vector. The number C_G is dependent on the sparsity of \mathbf{G} . If \mathbf{G} is dense, $C_G \sim nm$, and if \mathbf{G} is sparse, $C_G \sim \max(c, n)$. In GAT and I-RAT, **Step 1** costs $O(c)$, and in P-RAT, **Step 1** costs C_G . In **Step 2**, the QR decomposition costs $O(m^2 \times c)$. The SVD decomposition in **Step 3** costs $m^2 \times n$. The total cost is $2 \times m \times C_G + O(m^2(c+n))$. Hence, the algorithms cost $O(n \times c)$ for dense matrices and $O(n)$ for sparse ones, while a standard SVD algorithm requires $O(c^2 \times n)$. Hence an FAT algorithm is faster than the corresponding standard algorithm at least by one order. By the way, in FAT algorithms, O is for a small constant.

15.3.4 Matlab Code of FAT Algorithms

The first code includes various FAT algorithms, assuming that the positive semi-definite kernel is constructed. It finds the approximative leading eigenvalues and eigenvectors of the kernel.

```

function [Y, ev] = fat(G, k, m, mode, verb)
% FAT for computing DR data from a DR kernel.
% It approximately computes DR data from a DR kernel
% SYNTAX:
%   Y = FAT(G,M,K,MODE,VERB);
% INPUTS:
%   G: N x C matrix. If C=N and G is positive
%       semi-definite, the outputs approximate
%       leading eigenvalues and eigenvectors.
%       Otherwise, if N>C, the outputs approximate
%       leading singular values and the o.n.
%       basis of the principal k-subspace, and
%       if C > N, G is considered as a data matrix
%       with data dimension N, and the outputs
%       approximate the k principal components of
%       the date and their weights.
%   k: # of leading eigenvalues/eigenvectors.
%   m: dimension of project space (m>k).

```

```

% MODE: FAT mode. Three models are
% supported:
% 'IRAT' (or 'i'):
%     m columns are randomly selected to span the
%     approximative principal subspace.
% 'PRAT' (or 'p'):
%     A random projection is applied to obtaining an
%     m-subspace that approximates principal subspace.
% 'GAT' (or 'g'): m greedy columns are selected
%     to span wrapped-coordinates subspace.
% VERB: if true, display comments for processing.
% OUTPUT:
%     Y: N x k eigenvector matrix.
%     ev: leading k eigenvalues
% Example:
%     R = randn(1000, 50); G =R*R';
%     m=25; k=5;
%     [Y,ev] = FAT(G,m,k);
%
% Created by Jianzhong Wang, 2008
% Initialize inputs.
[n, c]=size(G);
if n > c
    warning(1,'row>col. Ensure object is col vector.');
else
    G = G';
    [n, c]=size(G);
end
if nargin<5
    verb = true;
end
if nargin<4
    mode = 'greedy';
end
if nargin<3
    m = k + 20;
end
% make FAT only if m<<c.
if 2*m < c;
    mode = lower(mode(1));
    % make approximative wrapped data subspace
    % using different methods.
    switch mode
        case 'i'
            if verb
                disp('Randomize interpolating method.');
            end
            % randomly select m columns.
            ind = randi(c, 1, c);

```

```

list =(1:c);
dind = list(ind < m+1);
Gm = G(:,dind);
case 'p'
    if verb
        disp('Randomize project method.');
    end
    % make a random m-subspace of Col(G).
    rm = randn(c, m);
    Gm = G*rm;
case 'g'
    if verb
        disp('Greedy method.');
    end
    T = sum(G.^2);
    [D, ind] = sort(T, 'descend');
    dind = ind(1:m);
    Gm = G(:,dind);
otherwise
    massage = ['-This mode is not supported.\n...
    '-Please use "i","p", or "g" mode.\n'];
    error(massage);
end
% Step 2. Construct data-wrap matrix.
[P, R] = qr(Gm,0);
G = G*P;
end
% Step 3. Make SVD on wrapped data space to
% obtain leading eigenvalues and eigenvectors.
[Y, ev] = svds(G, k);

```

The following algorithm is an FAT algorithm adopting Isomap kernel.

```

function [Y val]= fatisomap(X,ndims,options)
% FAT for Isomap kernel.
% Compute Isomap DR using FAT algorithm
%
% SYNTAX:
%   Y = FATISOMAP(X,ndims,options);
% INPUTS:
%   'X': D x N matrix, X(:,i) is the ith point.
%   ndims: Dimension of output data.
%   OPTIONS:
%   .epsilon: epsilon-neighbors.
%   .k: k-neighbors.
%   .mode: FAT mode.
%       "i" for random interpolation
%       "p" for random projection
%       "g" for greedy
%   .indim (>ndims): dimension of wrapped space

```

```

%      that approximates principal subspace
%      .verb: display the processing verbally.
% OUTPUT:
%      Y: d x N dimension-reduced data matrix.
%      val: d leading eigenvalues.
% Example: Reduce Swiss roll to 2-D.
%      N=1000;
%      tt = (3*pi/2)*(1+2*rand(1,N));
%      height = 21*rand(1,N);
%      X = [tt.*cos(tt); height; tt.*sin(tt)];
%      d=2;
%      options.k=10;
%      [Y,ev] = WRISOMAP(X, d, options);
%
% CALL: data_neighborhood.m
%        dijkstra_fast.dll or symdijkstra.m
%        fat.m
%
% Initialize options.
options.null=0;
if isfield(options, 'verb')
    verb = options.verb;
else
    verb = 1;
end
if isfield(options, 'mode')
    mode = lower(options.mode);
else
    mode = 'g';
end
if isfield(options, 'indim')
    m = options.indim;
else
    m = ndims + 15;
end
% Part1-Part 3 are same as in the
% standard Isomap DR algorithm.
% Part 1: Construct data-graph.
if verb
    disp('- Construct data graph');
end
options.symmetric = 1;
D2 = data_neighborhood(X,options);

% Part 2: Compute graph distance.
if verb
    disp('- Compute graph distance matrix');
end
N = size(D2,1);
D1 = sqrt(D2);

```

```
% Try to use .dll for faster performance.
if exist('perform_dijkstra_fast.dll','file')
    DG = perform_dijkstra_fast(D1,1:N);
else
    DG = symdijkstra(D1, 1:N);
end
% Part 3: Generate Isomap kernel
if verb
    disp('- Generate Isomap kernel');
end
DG = DG.^2;
GC = -.5*(DG - sum(DG) * ones(1,N)/N ...
    - ones(N,1)*sum(DG)/N + sum(DG(:))/(N^2));
% Part 4: Wrap-revealing approximation.
if verb
    disp('- Data-wrap revealing approximation');
end
[Y, val] = fat(GC, ndims, m, mode, verb);
Y = Y';
val = diag(val);
```

Remark 15.3. In the above code, we only use Gaussian-type random matrix in random projection. The other types of random matrices, say, Type 2 and Type 3 (see Section 7.2), can also be used in the algorithms.

15.4 Implementation of FAT Algorithms

In this section, we illustrate with various examples the validity and efficiency of the FAT algorithms.

15.4.1 FAT DR of Artificial Surfaces

All computations were carried out on a Mac Pro Desktop computer with 3.06 GHz Intel Core 2 Duo with 4 GB 1067 MHz DDR3 and Mac OS X version 10.5.8 operating system. The algorithms are run by Matlab.

The first four experiments display the results of various FAT algorithms performed on four surfaces: 3D-cluster, Punched sphere, Swiss roll, and S-curve. To carry out our experimentation, 2,000 points are randomly sampled on each surface. The surfaces are reduced to 2-dimensional data on the plane. We select Isomap Kernel in the experiments. The 10-neighborhood is selected for computing the graph metric on the data set.

In the following, we compare the efficiency as well as quality of the algorithms by compiling the CPU time and illustrating the three largest eigen-

values and deviations, as well as displaying the resulting DR pictures. Our experimental results on CPU time, three largest eigenvalues, and deviation from the standard Isomap method for the four methods, namely, standard Isomap method, GAT, I-RAT, and P-RAT (applied to the Isomap DR kernel) are compared in Tables 15.1–15.4; plots of the DR datasets on the plane are displayed in Figs. 15.1–15.4 (where Fig. 15.1 corresponds to Table 1, ⋯, and Fig. 15.4 corresponds to Table 15.4).

Table 15.1 Comparison of standard Isomap algorithm with FAT applied to Isomap DR kernel for Swiss roll: All FAT algorithms employ the 10-neighborhood and Gaussian random projection is used in PRAT. For normalization, all eigenvalues are divided by the first eigenvalue = 936340 of the DR kernel.

Algorithm	CPU time	eigen 1	eigen 2	eigen 3	deviation
Isomaps	2.5285	1.0000	0.0529	0.0093	
GAT	0.1523	1.000	0.0514	0.0072	0.0283
IRAT	0.1018	1.0000	0.0529	0.0091	0.0017
PRAT	0.1025	1.0000	0.0529	0.0089	0.0014

Table 15.2 Comparison of standard Isomap algorithm with FAT applied to Isomap DR kernel for S-curve: All FAT algorithms employ the 10-neighborhood and Gaussian random projection is used in PRAT. For normalization, all eigenvalues are divided by the first eigenvalue = 9985 of the DR kernel.

Algorithm	CPU time	eigen 1	eigen 2	eigen 3	deviation
Isomaps	2.7253	1.0000	0.3097	0.0280	
GAT	0.1519	1.0000	0.3094	0.0273	0.0014
IRAT	0.1036	1.0000	0.3096	0.0274	0.0001
PRAT	0.1144	1.0000	0.3096	0.0276	0.0002

Table 15.3 Comparison of standard Isomap algorithm with FAT applied to Isomap DR kernel for 3D-cluster: All FAT algorithms employ the 10-neighborhood and Gaussian random projection is used in PRAT. For normalization, all eigenvalues are divided by the first eigenvalue = 12900 of the DR kernel.

Algorithm	CPU time	eigen 1	eigen 2	eigen 3	deviation
Isomaps	2.9455	1.0000	0.0974	0.0053	
GAT	0.1542	0.9997	0.0792	0.0046	0.0088
IRAT	0.1203	1.0000	0.0974	0.0053	0.0000
PRAT	0.1287	1.0000	0.0974	0.0053	0.0001

Table 15.4 Comparison of standard Isomap algorithm with FAT applied to Isomap DR kernel for Punched sphere: All FAT algorithms employ the 10-neighborhood and Gaussian random projection is used in PRAT. For normalization, all eigenvalues are divided by the first eigenvalue = 716.63 of the DR kernel.

Algorithm	CPU time	eigen 1	eigen 2	eigen 3	deviation
Isomaps	1.4423	1.0000	0.9459	0.3852	
GAT	0.0954	1.0000	0.9105	0.3497	0.0543
IRAT	0.0636	1.0000	0.9455	0.3849	0.0020
PRAT	0.0739	1.0000	0.9457	0.3845	0.0010

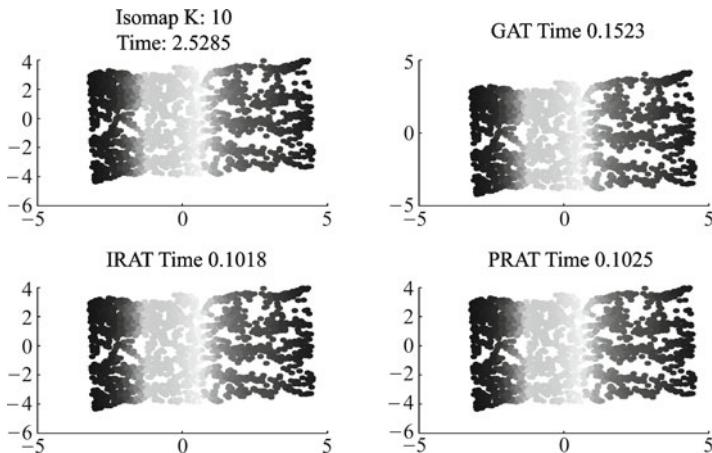


Fig. 15.1 Comparison of Isomaps with FAT algorithms applied to dataset of 2,000 points on Swiss roll. Top left: Standard Isomap. Top right: GAT. Bottom left: IRAT. Bottom right: PRAT.

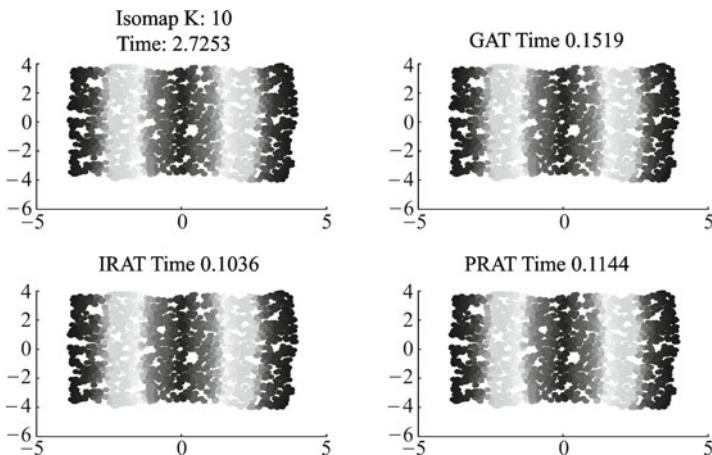


Fig. 15.2 Comparison of Isomaps with FAT algorithms applied to dataset of 2,000 points on S-curve. Top left: Standard Isomap. Top right: GAT. Bottom left: IRAT. Bottom right: PRAT.

Table 15.5 Comparison of standard Isomap algorithm with PRAT applied to Isomap DR kernel for S-curve: All PRAT algorithms employ the 10-neighborhood and random projections of three different types are applied. For normalization, all eigenvalues are divided by the first eigenvalue = 9985 of the DR kernel.

Algorithm	CPU time	eigen 1	eigen 2	eigen 3	deviation
Isomaps	2.7681	1.0000	0.3097	0.0280	
PRAT-1	0.0826	1.0000	0.3096	0.0277	0.0004
PRAT-2	0.0758	1.0000	0.3096	0.0277	0.0001
PRAT-3	0.0714	1.0000	0.3096	0.0274	0.0002

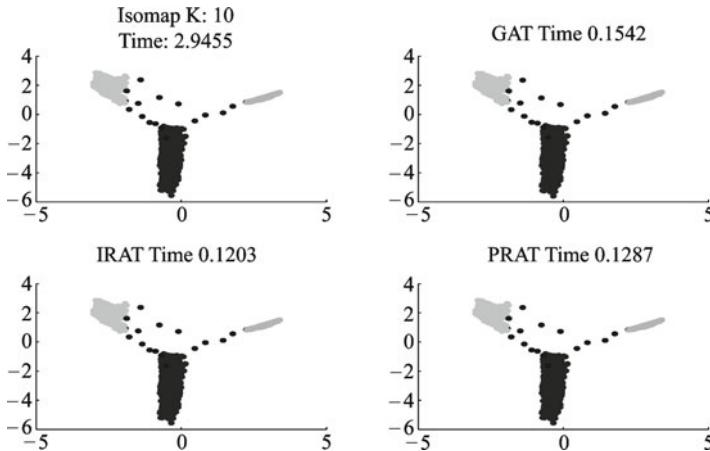


Fig. 15.3 Comparison of Isomaps with FAT algorithms applied to dataset of 2,000 points on 3D-cluster. Top left: Standard Isomaps. Top right: GAT. Bottom left: IRAT. Bottom right: PRAT.

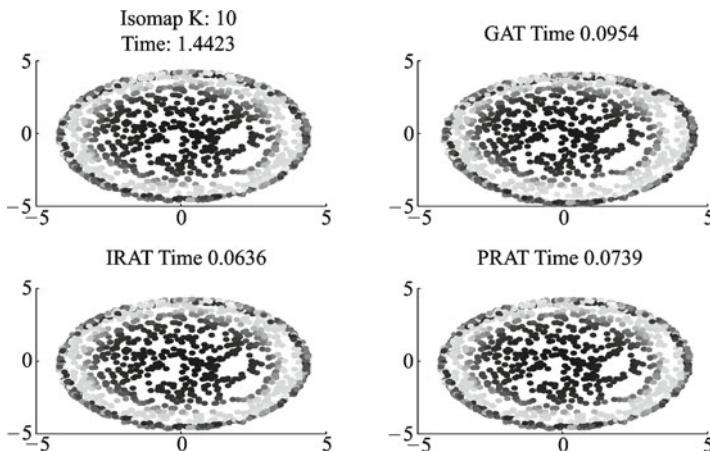


Fig. 15.4 Comparison of Isomaps with FAT algorithms applied to dataset of 2,000 points on Punched sphere. Top left: Standard Isomaps. Top right: GAT. Bottom left: IRAT. Bottom right: PRAT.

Table 15.6 Comparison of standard Isomap algorithm with PRAT applied to Isomap DR kernel for Punched sphere: All PRAT algorithms employ the 10-neighborhood and random projections of three different types are applied. For normalization, all eigenvalues are divided by the first eigenvalue = 725.76 of the DR kernel.

Algorithm	CPU time	eigen 1	eigen 2	eigen 3	deviation
Isomaps	1.3520	1.0000	0.9819	0.3870	
PRAT-1	0.0572	0.9998	0.9817	0.3867	0.0005
PRAT-2	0.0452	0.9997	0.9814	0.3859	0.0016
PRAT-3	0.0484	0.9998	0.9817	0.3865	0.0005

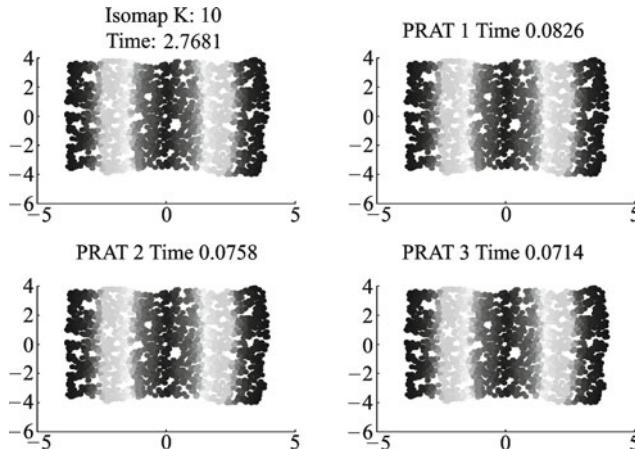


Fig. 15.5 Comparison of Isomaps with PRAT algorithms (types 1–3) applied to dataset of 2,000 points on S-curve. Top left: Standard Isomaps. Top right: PRAT of Type 1. Bottom left: PRAT of Type 2. Bottom right: PRAT of Type 3.

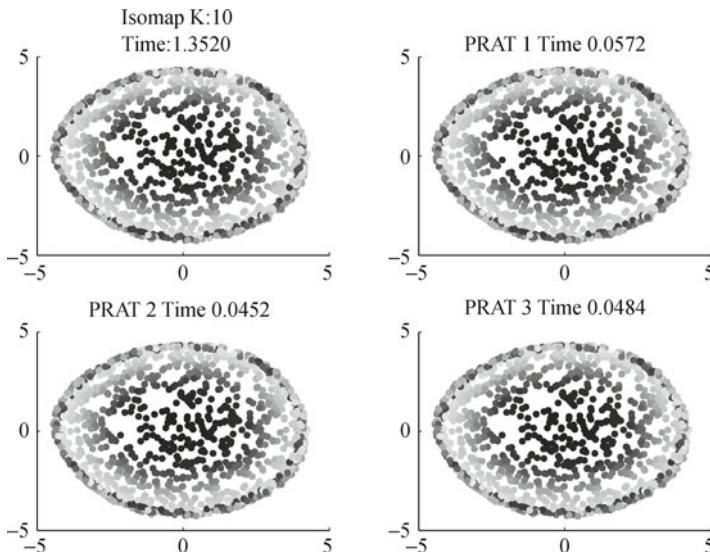


Fig. 15.6 Comparison of Isomaps with PRAT algorithms (types 1–3) applied to dataset of 2,000 points on Punched sphere. Top left: Standard Isomaps. Top right: PRAT of Type 1. Bottom left: PRAT of Type 2. Bottom right: PRAT of Type 3.

The deviation of FAT (applied to the Isomap DR kernel) from the standard Isomap method is defined as follows. Let $\mathbf{Y}_0 = [y_1^0 \ y_2^0]$ and $\mathbf{Y}_i = [y_1^i \ y_2^i]$ denote the matrices for the DR datasets obtained by applying the standard Isomap algorithm and the GAT, IRAT, DRAT algorithm, respectively. For the matrices \mathbf{Y}_0 and \mathbf{Y}_i , each column is normalized to be a unit vector

such that $\langle y_1^0, y_1^i \rangle > 0$ and $\langle y_2^0, y_2^i \rangle > 0$. Then the deviation is measured by $\|\mathbf{Y}_0 - \mathbf{Y}_i\|$.

In the first example, the four methods are compared in the DR of Swiss roll data. The Gaussian random projection (i.e. type-1) is used in PRAT. The experimental DR results are compiled in Table 15.1. Observe that all FAT algorithms are much more efficient than the standard Isomap algorithm, while the deviations of FAT algorithms from Isomap are negligible.

In Fig. 15.1, the plots corresponding to the DR results in Table 15.1 are displayed. The scattered plots to distinguish the datasets are color-coded. The plots show that all FAT algorithms perform very well.

In the second example, the four methods are compared in the DR of S-curve data. The Gaussian random projection (i.e. type-1) is used in PRAT. The experimental DR results are compiled in Table 15.2. Observe that all FAT algorithms are much more efficient than the standard Isomap algorithm, while the deviations of RAT from Isomap are again negligible.

In Fig. 15.2, the plots of all the DR results corresponding to Table 15.2 are displayed. Again scattered plots to distinguish the datasets are color-coded. The plots show that all FAT algorithms perform very well.

In the third example, the four methods are compared in the DR of 3D-cluster data. The Gaussian random projection (i.e. type-1) is used in PRAT. The experimental DR results are compiled in Table 15.3. Observe that all FAT algorithms are much more efficient than the standard Isomap algorithm, while the deviations of RAT from Isomap are again negligible. In Fig. 15.3, the plots of all the DR results listed in Table 15.3 are displayed. Again scattered plots to distinguish the datasets before and after DR are color-coded. The plots show that all FAT algorithms again perform very well.

In the fourth example, the four methods are compared in the DR of Punched sphere data. The Gaussian random projection (i.e. type-1) is used in PRAT. The experimental DR results are compiled in Table 15.4. Observe that all FAT algorithms are much more efficient than the standard Isomap algorithm, while the deviations of FAT algorithm from Isomap are negligible. In Fig. 15.4, the plots corresponding to the DR results in Table 15.4 are displayed. The plots show that all FAT algorithms perform very well.

In the fifth example of the S-curve data, random projections of three different types are used in PRAT algorithm and the experimental DR results are compiled in Table 15.5. Observe that all PRAT algorithms are again much more efficient than the standard Isomap algorithm, while the deviations of PRAT from Isomap are negligible. Also note that there is no noticeable difference between different PRAT algorithms in this experiment.

In Fig. 15.5, the plots of all the DR results listed in Table 15.5 are displayed. Again scattered plots to distinguish the datasets are color-coded. The plots show that the results are similar for random projection of all three types.

In the sixth example of the Punched sphere data, random projections of three different types are used in PRAT algorithm and the experimental DR results are compiled in Table 15.6. Observe that all PRAT algorithms

are again much more efficient than the standard Isomap algorithm, while the deviations of PRAT from Isomap are again negligible. Also note that there is no noticeable difference between different PRAT algorithms in this experiment.

In Fig. 15.6, the plots of all the DR results listed in Table 15.6 are displayed. Again scattered plots to distinguish the datasets are color-coded. The plots show that the results are similar for random projection of all three types.

15.4.2 Application of FAT to Sorting Image Datasets

In this experiment, we apply FAT algorithms to face recognition by sorting a given set of images. The data set \mathcal{X} for this experiment consists of 36 human-face images in different poses, and these images were randomly arranged. The resolution of each face image is 112×92 , and every image is considered as a point in $\mathbb{R}^{112 \times 92} = \mathbb{R}^{10,304}$. In other words, the given dataset can be written as $\mathcal{X} := \{\mathbf{x}_1, \dots, \mathbf{x}_{36}\} \subset \mathbb{R}^{10,304}$, with \mathbf{x}_i representing the i th human-face image. For the purpose of illustration, only the Isomap kernel is considered in this experiment. Since the objective of this experiment can be sorted, we consider $d = 1$, so that the DR data \mathcal{Y} are a set in \mathbb{R} ,

$$y_i = h(\mathbf{x}_i), \quad i = 1, \dots, 36,$$

with $\mathbf{Y} = [y_1 \ \dots \ y_{36}]$, where h maps $\mathbb{R}^{10,304}$ to \mathbb{R}^1 , reducing the data dimension from $m = 10,304$ to $d = 1$. In the experiment, we set the dimension of the wrapped coordinate subspace S at 21. Then the RAT algorithms in this experiment first create S by random projection and then project \mathcal{X} on S to produce the wrapped coordinate representation of \mathcal{X} , say, $\mathcal{P} = \{\mathbf{p}_1, \dots, \mathbf{p}_{36}\}$, such that

$$f(\mathbf{x}_i) = \mathbf{p}_i, \quad 1 \leq i \leq 36,$$

where f is the o.g. project from \mathbb{R}^n to S . Finally, the set \mathcal{Y} is obtained as the first principal component of \mathcal{P} . Sorting of the images in \mathcal{X} is accomplished simply by following the indices in the non-increasing order of the values of \mathcal{Y} , namely:

$$\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{36}\} \longrightarrow \{\mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \dots, \mathbf{x}_{i_{36}}\},$$

where the indices i_1, i_2, \dots, i_{36} are determined by $y_{i_1} \geq y_{i_2} \geq \dots \geq y_{i_{36}}$.

In this experiment, we applied both PRAT of Type 1 and PRAT of Type 2 algorithms to compare with the standard Isomap algorithm. By using the standard Isomap algorithm as well as PRAT 1 and PRAT 2, we obtained 3 pictures of one-dimensional sorting of the original data set \mathcal{X} of 36 human-face images, respectively. The results are displayed in Fig. 15.7, where four pictures of human-face images are presented. For each set, the 36 images are ordered from top to bottom and then left to right. The image set on the top-left of Fig. 15.7 is the original disordered one; the sets on the top-right,

the bottom-left, and the bottom-right are sorted by applying the standard Isomap algorithm, PRAT 1 and PRAT 2, respectively. The experimental result shows that all of them yield an identical sorting result.

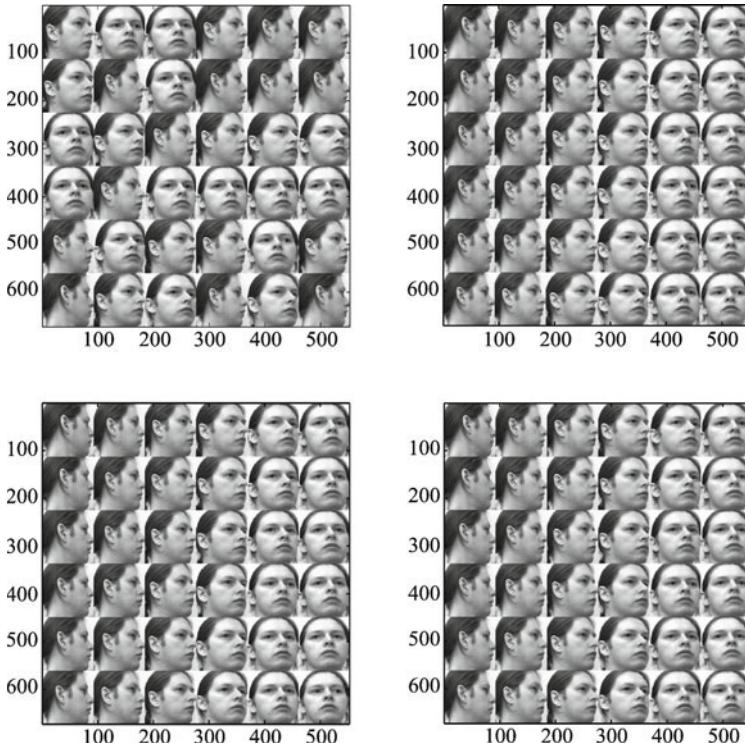


Fig. 15.7 Top left: Human faces in a random order. Top right: Faces sorted by Isomaps. Bottom left: Faces sorted by PRAT 1. Bottom right: Faces sorted by PRAT 2.

We remark that this experiment only uses 36 human-face images just because the pictures of the experiment can be clearly displayed. FAT algorithms can run very well for much larger image sets.

15.4.3 Conclusion

FAT applied to a nonlinear DR kernel reduces the dimensionality of the DR kernel by mapping the original data set to a wrapped coordinate space \mathbb{R}^m , in which an auxiliary data set, called wrapped coordinate representation of the data set, is created. FAT algorithms facilitate incorporating fast linear methods, such as PCA, with the nonlinear dimensionality reduction approach, by avoiding spectral decomposition of high-dimensional matrices and providing

faster and more stable computations. The mathematical and statistical analysis of FAT provides a theoretical foundation for the methods introduced. In FAT algorithms, RAT algorithms prove effective and extremely fast. Various experimental results in this chapter demonstrate the validity of the FAT algorithms.

15.5 Justification

In this section, we prove Theorems 15.7 and 15.8.

15.5.1 Main Proof of Theorem

For readers' convenience, we re-state the theorems here.

Theorem (15.7 and 15.8). *Let \mathbf{A} be an $m \times n$ matrix, $k \leq l < \min(n, m)$. Let the triple $[\mathbf{U}, \Sigma, \mathbf{V}]$ be produced by the randomized algorithm of low-rank SVD approximation. Then*

$$\|\mathbf{A} - \mathbf{U}\Sigma\mathbf{V}'\| \leq 2 \left(\sqrt{2lm\beta\gamma + 1} + \sqrt{2lm\beta\gamma} \right) \sigma_{k+1} \quad (15.49)$$

with the probability at least χ , which is given in (15.32).

Furthermore, let $j > 0$ satisfy $k + j < \min(m, n)$. Write $m_1 = \max(m - k - j, l)$, $m_2 = \max(j, l)$, and $m_3 = \max(j + k, l)$. Then

$$\|\mathbf{A} - \mathbf{U}\Sigma\mathbf{V}'\| \leq \xi \sigma_{k+1} + \eta \sigma_{k+j+1} \quad (15.50)$$

with the probability at least κ , which is given in (15.34), where

$$\xi = 2 \left(\sqrt{2l\beta\gamma m_2 + 1} + \sqrt{2l\beta\gamma m_3} \right), \quad (15.51)$$

$$\eta = 2 \left(\sqrt{2l\beta\gamma m_1 + 1} + \sqrt{2l\beta\gamma m_1} \right). \quad (15.52)$$

We now start the proof of the Theorem.

Proof. The main idea of the proof is the following. As mentioned in the description of the randomized SVD algorithm, since $\text{Row}(\mathbf{R})$ approximates the principal k -subspace of $\text{Row}(\mathbf{A})$, there is an $m \times k$ matrix \mathbf{F} such that $\mathbf{FR} \approx \mathbf{A}$. Before we construct \mathbf{F} , we first discuss the error of $\mathbf{A} - \mathbf{FR}$ in general. Let \mathbf{Q} be the matrix obtained in Step 2, whose columns form an o.n. basis of $\text{Row}(\mathbf{R})$. We have

$$\begin{aligned} \mathbf{A} - \mathbf{U}\Sigma\mathbf{V}' &= \mathbf{A} - \mathbf{U}\Sigma\mathbf{W}'\mathbf{Q}' \\ &= \mathbf{A} - \mathbf{AQ}\mathbf{Q}' \end{aligned}$$

and

$$\mathbf{A} - \mathbf{AQ}\mathbf{Q}' = \mathbf{A} - \mathbf{FR} + \mathbf{FR} - \mathbf{FS}'\mathbf{Q}' + \mathbf{FS}'\mathbf{Q}' - \mathbf{AQ}\mathbf{Q}', \quad (15.53)$$

where

$$\begin{aligned} \mathbf{F}\mathbf{S}'\mathbf{Q}' - \mathbf{A}\mathbf{Q}\mathbf{Q}' &= (\mathbf{F}\mathbf{S}'\mathbf{Q}' - \mathbf{A})\mathbf{Q}\mathbf{Q}' \\ &= (\mathbf{F}\mathbf{S}'\mathbf{Q}' - \mathbf{F}\mathbf{R} + \mathbf{F}\mathbf{R} - \mathbf{A})\mathbf{Q}\mathbf{Q}'. \end{aligned}$$

Hence,

$$\mathbf{A} - \mathbf{A}\mathbf{Q}\mathbf{Q}' = (\mathbf{A} - \mathbf{F}\mathbf{R} + \mathbf{F}\mathbf{R} - \mathbf{F}\mathbf{S}'\mathbf{Q}')(\mathbf{I} - \mathbf{Q}\mathbf{Q}').$$

By $\|\mathbf{Q}\mathbf{Q}'\| = 1$ and $\mathbf{R} = \mathbf{G}\mathbf{A}$,

$$\begin{aligned} \|\mathbf{A} - \mathbf{U}\Sigma\mathbf{V}'\| &= \|\mathbf{A} - \mathbf{A}\mathbf{Q}\mathbf{Q}'\| \\ &\leq (\|\mathbf{A} - \mathbf{F}\mathbf{R}\| + \|\mathbf{F}\mathbf{R} - \mathbf{F}\mathbf{S}'\mathbf{Q}'\|)\|\mathbf{I} - \mathbf{Q}\mathbf{Q}'\| \\ &\leq 2(\|\mathbf{A} - \mathbf{F}\mathbf{G}\mathbf{A}\| + \|\mathbf{F}\|\|\mathbf{Q}\mathbf{S} - \mathbf{R}'\|). \end{aligned}$$

By Lemma 15.3 in the next subsection, we have

$$\|\mathbf{A} - \mathbf{F}\mathbf{G}\mathbf{A}\| \leq \sqrt{2lm\beta\gamma + 1}\sigma_{k+1} \quad (15.54)$$

with the probability $1 - \tilde{p}(l, k, \beta) - p(m, \gamma)$. By Lemma 15.4 in the next subsection, we have

$$\|\mathbf{F}\|\|\mathbf{Q}\mathbf{S} - \mathbf{R}'\| \leq \sqrt{2lm\beta\gamma}\sigma_{k+1} \quad (15.55)$$

with the probability $1 - \tilde{p}(n, k, \beta)$. Then, (15.49) is proved. Similarly, by Lemmas 15.4 and 16.4 in the next subsection, we obtain (15.50). The theorem is proved. \square

15.5.2 Lemmas Used in the Proof

The following lemma gives the estimation of the spectrum radius of a Gaussian random matrix [25, 31].

Lemma 15.1. *Suppose that $l, j, m \in \mathbb{N}$ with $m \geq \max(l, j)$. Let σ_{\max} be the greatest singular value of an $l \times j$ random matrix \mathbf{G} whose entries are i.i.d. Gaussian random variables of zero mean and unit variance. Then*

$$\Pr(\sigma_{\max} \leq \sqrt{2m\gamma}) \geq 1 - p(m, \gamma), \quad (15.56)$$

where $\gamma > 1$ is a real number making the right-hand side of (15.56) positive, and $p(x, \gamma)$ is defined in (15.30).

A lower bound on the least singular value of a random matrix is given in the following lemma.

Lemma 15.2 [32]. *Suppose that $l, k \in \mathbb{N}$ with $l \geq k$ and \mathbf{G} is an $l \times k$ random matrix whose entries are i.i.d. Gaussian random variables of zero mean and unit variance. Let σ_{\min} be the least singular value of \mathbf{G} . Then*

$$\Pr\left(\sigma_{\min} \geq \frac{1}{\sqrt{l\beta}}\right) \geq 1 - \tilde{p}(l, k, \beta), \quad (15.57)$$

where $\beta > 0$ is a real number making the right-hand side of (15.57) positive, and $\tilde{p}(l, k, \beta)$ is given in (15.29).

Lemma 15.2 claims that a random matrix almost surely has full rank. We say an event to be *almost sure* if its probability is $1 - \varepsilon$ for any sufficiently small $\varepsilon > 0$.

To find the recovering matrix \mathbf{F} in the theorem, we need the following.

Lemma 15.3. *Let k, l, n be positive integers with $k \leq l < n$. Let $\mathbf{A} \in \mathfrak{M}_{m,n}$ and \mathbf{G} be an $l \times m$ random matrix whose entries are i.i.d. Gaussian random variables of zero mean and unit variance, and let $\gamma > 1$ and $\beta > 0$ be chosen as in Lemmas 15.1 and 15.2 respectively. Let σ_{k+1} be the $(k+1)$ st largest singular value of \mathbf{A} . Then there is an $m \times l$ matrix \mathbf{F} such that*

$$\|\mathbf{FGA} - \mathbf{A}\| \leq \sigma_{k+1} \sqrt{2lm\beta\gamma + 1} \quad (15.58)$$

with the probability $1 - \tilde{p}(l, k, \beta) - p(m, \gamma)$, and

$$\|\mathbf{F}\| \leq \sqrt{l}\beta \quad (15.59)$$

with the probability no less than $1 - \tilde{p}(l, k, \beta)$.

Proof. Assume that the singular value decomposition of \mathbf{A} is

$$\mathbf{A} = \mathbf{U} \boldsymbol{\Sigma}_{\mathbf{A}} \mathbf{V}',$$

where $\mathbf{U} \in \mathfrak{O}_m$, $\boldsymbol{\Sigma}_{\mathbf{A}}$ is an $m \times n$ diagonal matrix whose diagonal entries are the singular values of \mathbf{A} : $\sigma_1, \dots, \sigma_{\min(m,n)}$, and $\mathbf{V} \in \mathfrak{O}_n$. Because \mathbf{G} is i.i.d. Gaussian and \mathbf{U} is o.g., the matrix

$$\tilde{\mathbf{G}} = \mathbf{GU}$$

is an i.i.d. Gaussian random matrix. We write

$$\tilde{\mathbf{G}} = [\mathbf{G}_1 \mid \mathbf{G}_2],$$

where $\mathbf{G}_1 \in \mathfrak{R}_{l,k}$ and $\mathbf{G}_2 \in \mathfrak{R}_{l,(m-k)}$. Denoting by \mathbf{G}_1^- the left inverse of \mathbf{G}_1 , we have, by Lemma 15.2,

$$\|\mathbf{G}_1^-\|_2 = 1/\sigma_{\min}(\mathbf{G}_1) \leq \sqrt{l}\beta \quad (15.60)$$

with the probability $1 - \tilde{p}(l, k, \beta)$. Write

$$\mathbf{F} = \mathbf{U} \begin{bmatrix} \mathbf{G}_1^- \\ 0 \end{bmatrix}.$$

Since \mathbf{U} is o.g., \mathbf{F} has the same spectrum as \mathbf{G}_1^- . Hence,

$$\|\mathbf{F}\| \leq \sqrt{l}\beta \quad (15.61)$$

with the probability $1 - \tilde{p}(l, k, \beta)$. It is also followed that

$$\mathbf{F}\tilde{\mathbf{G}} = \begin{bmatrix} \mathbf{I} & \mathbf{G}_1^-\mathbf{G}_2 \\ 0 & 0 \end{bmatrix} \mathbf{V}',$$

which yields

$$\begin{aligned} (\mathbf{F}\mathbf{G})\mathbf{A} &= \mathbf{U}\mathbf{F}\mathbf{G}\mathbf{U}\boldsymbol{\Sigma}_A(\mathbf{V}') = \mathbf{U}\mathbf{F}\tilde{\mathbf{G}}\boldsymbol{\Sigma}_A\tilde{\mathbf{G}} \\ &= \mathbf{U} \begin{bmatrix} \mathbf{I} & \mathbf{G}_1^- \mathbf{G}_2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_1 & 0 \\ 0 & \boldsymbol{\Sigma}_2 \end{bmatrix} \mathbf{V}' \\ &= \mathbf{U} \begin{bmatrix} \boldsymbol{\Sigma}_1 & \boldsymbol{\Sigma}_2 \mathbf{G}_1^- \mathbf{G}_2 \\ 0 & 0 \end{bmatrix} \mathbf{V}', \end{aligned}$$

where $\boldsymbol{\Sigma}_1 = \text{diag}(\sigma_1, \dots, \sigma_k)$ and $\|\boldsymbol{\Sigma}_2\| = \sigma_{k+1}$. Hence,

$$\mathbf{F}\mathbf{G}\mathbf{A} - \mathbf{A} = \mathbf{U} \begin{bmatrix} 0 & \boldsymbol{\Sigma}_2 \mathbf{G}_1^- \mathbf{G}_2 \\ 0 & -\boldsymbol{\Sigma}_2 \end{bmatrix} \mathbf{V}',$$

which yields

$$\|\mathbf{A}\mathbf{G}\mathbf{H} - \mathbf{A}\| \leq \sqrt{(\|\mathbf{G}_1^-\| \|\mathbf{G}_2\|)^2 + 1\sigma_{k+1}}. \quad (15.62)$$

By Lemma 15.1, for $m \geq \max(l, n - k)$,

$$\|\mathbf{G}_2\|_S \leq \sqrt{2m\gamma} \quad (15.63)$$

with the probability $1 - p(m, \gamma)$. Combining (15.60), (15.63), (15.61), and (16.62), we have (15.58) and (15.59). The lemma is proved. \square

Lemma 15.4. *Let \mathbf{G} be the random matrix in Lemma 15.3, \mathbf{Q} and \mathbf{R} be the matrices created in Step 2, such that*

$$\|\mathbf{G}\mathbf{A} - \mathbf{S}'\mathbf{Q}'\| \leq \rho_{k+1}, \quad (15.64)$$

where ρ_{k+1} is the $(k+1)$ st greatest singular value of $\mathbf{R} = \mathbf{G}\mathbf{A}$. Let σ_{k+1} be the $(k+1)$ st greatest singular value of \mathbf{A} . Then

$$\|\mathbf{F}\mathbf{G}\mathbf{A} - \mathbf{F}\mathbf{S}'\mathbf{Q}'\| \leq \sqrt{2lm\beta\gamma}\sigma_{k+1} \quad (15.65)$$

with the probability $1 - \tilde{p}(n, k, \beta) - p(m, \gamma)$.

Proof. For any vector $\mathbf{v} \in \mathbb{R}^n$ with $\mathbf{v} \neq 0$,

$$\frac{\|\mathbf{G}\mathbf{A}\mathbf{v}\|}{\|\mathbf{v}\|} \leq \|\mathbf{G}\| \frac{\|\mathbf{A}\mathbf{v}\|}{\|\mathbf{v}\|}, \quad (15.66)$$

which yields

$$\rho_{k+1} \leq \|\mathbf{G}\|\sigma_{k+1}. \quad (15.67)$$

By Lemma 15.1, $\|\mathbf{G}\| \leq \sqrt{2m\gamma}$ with the probability $1 - p(m, \gamma)$. The equation (15.67), together with (15.61), yields (15.65). The lemma is proved. \square

To obtain the estimate of (15.50), we need the following.

Lemma 15.5. *Let \mathbf{A} and \mathbf{G} be given as in Lemma 15.3. Write $m_1 = \max(m - k - j, l)$ and $m_2 = \max(j, l)$. Then for any positive integer j with $k + j < \min(n, m)$, there is a matrix $\mathbf{F} \in \mathfrak{M}_{l,m}$ such that*

$$\|\mathbf{A} - \mathbf{FGA}\| \leq \left(\sigma_{k+1} \sqrt{2lm_2\beta\gamma + 1} + \sigma_{k+j+1} \sqrt{2lm_1\beta\gamma} \right) \quad (15.68)$$

with the probability no less than κ .

Since the proof of Lemma 15.5 is quite similar to the proof of Lemma 15.3, we skip it here.

References

- [1] Cheng, H., Gimbutas, Z., Martinsson, P.G., Rokhlin, V.: On the compression of low rank matrices. SIAM Journal on Scientific Computing 26(4), 1389–1404 (2005).
- [2] Woolfe, F., Liberty, E., Rokhlin, V., Tygert, M.: A randomized algorithm for the approximation of matrices. Appl. Comput. Harmon. Anal. 25(3), 335–366 (2008).
- [3] Hansen, P.C.: Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion. SIAM, Philadelphia (1998).
- [4] Stewart, G.W.: Matrix Algorithms Volume I: Basic Decompositions. SIAM, Philadelphia (1998).
- [5] Chan, T.F., Hansen, P.C.: Some applications of the rank revealing QR factorization. SIAM J. Sci. Statist. Comput. 13, 727–741 (1992).
- [6] Gu, M., Eisenstat, S.C.: Efficient algorithms for computing a strong rank-revealing QR factorization. SIAM J. Sci. Comput. 17, 848–869 (1996).
- [7] Hong, Y.P., Pan, C.T.: Rank-revealing QR factorizations and the singular value decomposition. Mathematics of Computation 58(197), 213–232 (1992).
- [8] Berry, M., Pulatova, S., Stewart, G.: Algorithm 844: computing sparse reduced-rank approximations to sparse matrices. ACM Trans Math Softw 31(2), 252–269 (2005).
- [9] Goreinov, S.A., Tyrtyshnikov, E.E., Zamarashkin, N.L.: A theory of pseudo-skeleton approximations. Linear Algebra and Its Applications 261, 1–21 (1997).
- [10] Tyrtyshnikov, E.: Matrix bruhat decompositions with a remark on the QR (GR) algorithm. Linear Algebra Appl. 250, 61–68 (1997).
- [11] Tyrtyshnikov, E., Zamarashkin, N.: Thin structure of eigenvalue clusters for non-hermitian Toeplitz matrices. Linear Algebra Appl. 292, 297–310 (1999).
- [12] Zamarashkin, N., Tyrtyshnikov, E.: Eigenvalue estimates for Hankel matrices. Sbornik: Mathematics 192, 59–72 (2001).
- [13] Fierro, R., Bunch, J.: Bounding the subspaces from rank revealing two-sided orthogonal decompositions. SIAM Matrix Anal. Appl. 16, 743–759 (1995).
- [14] Fierro, R., Hansen, P.: Low-rank revealing UTV decompositions. Numerical Algorithms 15, 37–55 (1997).
- [15] Fierro, R., Hansen, P.C., Hansen, P.S.K.: UTV Tools: Matlab templates for rank-revealing UTV decompositions. Numerical Algorithms 20, 165–194 (1999).

- [16] Golub, G.H., van Loan, C.F.: Matrix Computations, third edn. Johns Hopkins Press, Baltimore (1996).
- [17] Fierro, R., Hansen, P.: UTV Expansion Pack: Special-purpose rank revealing algorithms. Numerical Algorithms 40, 47–66 (2005).
- [18] Hansen, P.C., Yalamov, P.Y.: Computing symmetric rank-revealing decompositions via triangular factorization. SIAM J. Matrix Anal. Appl. 28, 443–458 (2001).
- [19] Luk, F.T., S, Q.: A symmetric rank-revealing Toeplitz matrix decomposition. J. VLSI Signal Proc. 14, 19–28 (1996).
- [20] Belabbas, M.A., Wolfe, P.J.: Fast low-rank approximation for covariance matrices. In: Proceedings of the 2nd IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (2007).
- [21] Belabbas, M.A., Wolfe, P.J.: On sparse representations of linear operators and the approximation of matrix products. In: Proceedings of the 42nd Annual Conference on Information Sciences and Systems, pp. 258–263 (2008).
- [22] Fowlkes, C., Belongie, S., Chung, F., Malik, J.: Spectral grouping using the Nyström method. IEEE Trans. Patt. Anal. Mach. Intell. pp. 214–225 (2004).
- [23] Parker, P., Wolfe, P.J., Tarokh, V.: A signal processing application of randomized low-rank approximations. In: IEEE Worksh. Statist. Signal Process., pp. 345–350 (2005).
- [24] Williams, C.K.I., Seeger, M.: Using the Nyström method to speed up kernel machines. In: Neural Information Processing Systems, pp. 682–688 (2000).
- [25] Martinsson, P.G., Rokhlin, V., Tygert, M.: A randomized algorithm for the approximation of matrices. Tech. Rep. 1361, Dept. of Computer Science, Yale University (2006).
- [26] Martinsson, P.G., Rokhlin, V., Tygert, M.: On interpolation and integration in finite-dimensional spaces of bounded functions. Comm. Appl. Math. Comput. Sci. pp. 133–142 (2006).
- [27] Belabbas, M.A., Wolfe, P.J.: Spectral methods in machine learning: New strategies for very large data sets. PANS 106(2), 369–374 (2009).
- [28] Chui, C., Wang, J.: Dimensionality reduction of hyper-spectral imagery data for feature classification. In: W. Freeden, Z. Nashed, T. Sonar (eds.) Handbook of Geomathematics. Springer, Berlin (2010).
- [29] Chui, C., Wang, J.: Randomized anisotropic transform for nonlinear dimensionality reduction. International Journal on Geomathematics 1(1), 23–50 (2010).
- [30] Xiao, L., Sun, J., Boyd, S.P.: A duality view of spectral methods for dimensionality reduction. In: W.W. Cohen, A. Moore (eds.) Machine Learning: Proceedings of the Twenty-Third International Conference, ACM International Conference Proceeding Series, vol. 148, pp. 1041–1048. ICML, Pittsburgh, Pennsylvania, USA (2006).
- [31] Goldstine, H.H., von Neumann, J.: Numerical inverting of matrices of high order II. Amer. Math. Soc. Proc. 2, 188–202 (1951).
- [32] Chen, Z., Dongarra, J.J.: Condition numbers of Gaussian random matrices. SIAM J. on Matrix Anal. Appl. 27, 603–620 (2005).

Appendix A Differential Forms and Operators on Manifolds

In this appendix, we introduce differential forms on manifolds so that the integration on manifolds can be treated in a more general framework. The operators on manifolds can be also discussed using differential forms. The material in this appendix can be used as a supplement of Chapter 2.

A.1 Differential Forms on Manifolds

Let M be a k -manifold and $\mathbf{u} = [u^1, \dots, u^k]$ be a coordinate system on M . A differential form of degree 1 in terms of the coordinates u has the expression

$$\alpha = \sum_{i=1}^k a_i du^i = (\mathbf{d}\mathbf{u})\mathbf{a},$$

where $\mathbf{d}\mathbf{u} = [du^1, \dots, du^k]$ and $\mathbf{a} = [a_1, \dots, a_k]' \in C^1$. The differential form is independent of the chosen coordinates. For instance, assume that $\mathbf{v} = [v^1, \dots, v^k]$ is another coordinate system and α in terms of the coordinates v has the expression

$$\alpha = \sum_{i=1}^n b_i dv^i = (\mathbf{d}\mathbf{v})\mathbf{b},$$

where $\mathbf{b} = [b_1, \dots, b_k]' \in C^1$. Write $\frac{\partial v}{\partial u} = \left[\frac{\partial v^i}{\partial u^j} \right]_{i,j=1}^n$. By

$$\mathbf{d}\mathbf{v} = \mathbf{d}\mathbf{u} \left(\frac{\partial v}{\partial u} \right)^T,$$

we have

$$\mathbf{d}\mathbf{v}\mathbf{b} = \mathbf{d}\mathbf{u} \left(\frac{\partial v}{\partial u} \right)^T \mathbf{b},$$

and by

$$\mathbf{a} = \left(\frac{\partial v}{\partial u} \right)^T \mathbf{b},$$

we have $(du)\mathbf{a} = (dv)\mathbf{b}$. From differential forms of degree 1, we inductively define exterior differential forms of higher degree using wedge product, which obeys usual associate law, distributive law, as well as the following laws:

$$\begin{aligned} du^i \wedge du^i &= 0, \\ du^i \wedge du^j &= -du^j \wedge du^i, \\ a \wedge du^i &= du^i \wedge a = adu^i, \end{aligned}$$

where $a \in \mathbb{R}$ is a scalar.

It is clear that on a k -dimensional manifold, all forms of degree greater than k vanish, and a form of degree k in terms of the coordinates u has the expression

$$\omega = a_{12\dots k} du^1 \wedge \dots \wedge du^k, \quad a_{12\dots k} \in C^1. \quad (\text{A.1})$$

To move differential forms from one manifold to another, we introduce the inverse image (also called pullback) of a function f . Let M_1 and M_2 be two manifolds, with the coordinate systems u and v respectively. Let $\mu \in C^r$ ($r \geq 1$) be a mapping from M_2 into M_1 , $\mu(\mathbf{y}) = \mathbf{x}$, $\mathbf{y} \in M_2$. Let g be the parameterization of M_2 : $\mathbf{y} = g(v)$. We have

$$u = (\mu \circ g)(v).$$

Hence, u is a C^r function of v . Let f be a function on M_1 . Then there is a function h on M_2 such that

$$h(\mathbf{y}) = f(\mu(\mathbf{y})), \quad \mathbf{y} \in M_2.$$

We denote the function h by $\mu^* f$ and call it the inverse image of f (by μ). Clearly, the inverse image has the following properties:

1. $\mu^*(f + g) = \mu^* f + \mu^* g$;
2. $\mu^*(fg) = (\mu^* f)(\mu^* g)$, $f \in C^1$;
3. if $df = \sum_i a_i du^i$, then

$$d(\mu^* f) = \sum_i (\mu^* a_i) d(\mu^* u^i).$$

We now define the inverse image of a differential form as follows.

Definition A.1. Let ω be a differential form of degree j :

$$\omega = f du^{n_1} \wedge \dots \wedge du^{n_j}.$$

Then the inverse image of the differential form ω (by μ) is defined by

$$\mu^* \omega = (\mu^* f) d(\mu^* u^{n_1}) \wedge \dots \wedge d(\mu^* u^{n_j}).$$

The properties of inverse image for functions therefore also hold for differential forms. Besides, by Definition A.1, the following properties hold.

$$\text{supp}(\mu^*\omega) \subset \mu(\text{supp}(\omega))$$

and

$$\mu^*d\omega = d(\mu^*\omega), \quad \mu^*(\omega_1 \wedge \omega_2) = \mu^*\omega_1 \wedge \mu^*\omega_2.$$

A.2 Integral over Manifold

Let M be a compact k -manifold having a single coordinate system (M, u) and ω be a k -form on M given in (A.1). Let $D = u(M) \subset \mathbb{R}^k$. Then the integral of ω over M is defined by

$$\int_M \omega = \int_D a_{12\dots k} du^1 \wedge \dots \wedge du^k.$$

We often simply denote $\int_M \omega$ by $\int \omega$ if no confusion arises. Notice that if the support of $a_{12\dots k}$ is a subset of D , then we have

$$\int_D a_{12\dots k} du^1 \wedge \dots \wedge du^k = \int_{\mathbb{R}^k} a_{12\dots k} du^1 \wedge \dots \wedge du^k.$$

To extend the definition of the integral over a manifold, which does not have a global coordinate system, we introduce the partition of unity and the orientation of a manifold.

Definition A.2. Assume that $\{U_i\}$ is an open covering of a manifold M . A countable set of functions $\{\phi_j\}$ is called a partition of unity of M with respect to the open covering $\{U_i\}$ if it satisfies the following conditions:

1. $\phi_j \in C^\infty$ has compact support contained in one of the U_i ;
2. $\phi_j \geq 0$ and $\sum_j \phi_j = 1$;
3. each $p \in M$ is covered by only a finite number of $\text{supp}(\phi_j)$.

It is known that each manifold has a partition of unity.

We shall use partition of unity to study orientation of manifolds. Orientation is an important topological property of a manifold. Some manifolds are oriented, others are not. For example, in \mathbb{R}^3 the Möbius strip is a non-oriented 2-dimensional manifold, while the sphere S^2 is an oriented one. To define the orientation of a manifold, we introduce the orientation of a vector space.

Definition A.3. Let $\mathcal{E} = \{e_1, \dots, e_n\}$ and $\mathcal{B} = \{b_1, \dots, b_n\}$ be two bases of the space \mathbb{R}^n . Let \mathbf{A} be a matrix, which represents the linear transformation from \mathcal{B} to \mathcal{E} such that

$$e_i = \mathbf{A}b_i, \quad i = 1, 2, \dots, n.$$

We say that \mathcal{E} and \mathcal{B} have the same orientation if $\det(\mathbf{A}) > 0$, and that they have the opposite orientations if $\det(\mathbf{A}) < 0$. Therefore, for each $n \in \mathbb{Z}_+$, \mathbb{R}^n has exactly two orientations, in which the orientation determined by the canonical o.n. basis is called the right-hand orientation, and the other is called the left-hand orientation.

The closed k -dimensional half-space is defined by

$$\mathbf{H}^k = \{[x_1, \dots, x_k]' \in \mathbb{R}^k : x_k \geq 0\}. \quad (\text{A.2})$$

Definition A.4. A connected k -dimensional manifold M ($k \geq 1$) is said to be oriented if, for each point in M , there is a neighborhood $W \subset M$ and a diffeomorphism h from W to \mathbb{R}^k or \mathbb{H}^k such that for each $\mathbf{x} \in W$, $dh_{\mathbf{x}}$ maps the given orientation of the tangent space $T_{\mathbf{x}}M$ to the right-hand orientation of \mathbb{R}^k .

By the definition, if M is a connected oriented manifold, there is a differential structure $\{(W_i, h_i)\}$ such that at each $\mathbf{p} \in h_i(W_i) \cap h_j(W_j)$, the linear transformation $d(h_i \circ h_j^{-1})_{\mathbf{p}} : \mathbb{R}^k \rightarrow \mathbb{R}^k$, satisfies $\det(d(h_i \circ h_j^{-1})_{\mathbf{p}}) > 0$ for each \mathbf{p} . We give two examples below to demonstrate the concept of orientation.

Example A.1. Let $\mathbf{U} = [-\pi, \pi] \times (-1, 1)$. Möbius strip \mathbb{M}^2 is given by the parametric equation $g : \mathbf{U} \rightarrow \mathbb{R}^3$:

$$g(u, v) = \left[\left(2 + v \sin\left(\frac{u}{2}\right) \right) \cos u, \left(2 + v \sin\left(\frac{u}{2}\right) \right) \sin u, v \cos\left(\frac{u}{2}\right) \right]^T. \quad (\text{A.3})$$

However, g is not a diffeomorphism from \mathbf{U} to $g(\mathbf{U}) \subset \mathbb{R}^3$ since it is not one-to-one (on the line $u = \pi$). To get an atlas on \mathbb{M}^2 , we define

$$\tilde{g}(u, v) = \left[\left(-2 + v \cos\left(\frac{u}{2}\right) \right) \cos u, \left(-2 + v \cos\left(\frac{u}{2}\right) \right) \sin u, v \sin\left(\frac{u}{2}\right) \right]^T, \quad (\text{A.4})$$

and set $U^o = (-\pi, \pi) \times (-1, 1)$. Then $\{(g(U^o), g^{-1}), (\tilde{g}(U^o), \tilde{g}^{-1})\}$ is a differentiable structure on \mathbb{M}^2 with $g(U^o) \cup \tilde{g}(U^o) = \mathbb{M}^2$. Note that the following lines

$$\begin{aligned} \mathbf{L}_1 &= \{[2, 0, t]^T : -1 < t < 1\}, \\ \mathbf{L}_2 &= \{[t, 0, 0]^T : -3 < t < -1\}, \end{aligned}$$

do not reside on $g(U^o) \cap \tilde{g}(U^o)$.

We compute the determinant $\det(d(\tilde{g}^{-1} \circ g))$ on different arcs in U^o :

$$\det(d(\tilde{g}^{-1} \circ g)) = \begin{cases} 1 & (u, v) \in (0, \pi) \times (-1, 1) \\ -1 & (u, v) \in (-\pi, 0) \times (-1, 1) \end{cases}.$$

The opposite signs show that the Möbius strip \mathbb{M}^2 is not oriented.

Example A.2. The spherical surface \mathbb{S}^2 defined in Example 2.1 is oriented. Recall that $\{(W_1, h_1), (W_2, h_2)\}$ is an atlas on \mathbb{S}^2 . The transformation from

$h_2 = (\tilde{u}, \tilde{v})$ to $h_1 = (u, v)$ is given in (2.25). The Jacobian of the transformation is negative everywhere, for we have

$$\det(df_{(\tilde{u}, \tilde{v})}) = -\frac{1}{(\tilde{u}^2 + \tilde{v}^2)^2} < 0.$$

Hence, \mathbb{S}^2 is oriented. Since $\det(df_{(\tilde{u}, \tilde{v})}) < 0$, in order to obtain the same orientation over both W_1 and W_2 , we can replace h_2 by $-h_2$.

The definition of integrable functions on a manifold M needs the concept of zero-measure set on manifold. To avoid being involved in the measure theory, we define the integration over a manifold for continuous functions only.

Definition A.5. Let M be a connected and oriented k -dimensional manifold, and $\Phi = \{\phi_j\}_{j \in \Gamma}$ be a partition of unity of M so that the support of each $\phi_j \in \Phi$ is contained in the domain of a coordinate system. Let ω be a continuous differential form of degree k on M . The integral of ω over M is defined as

$$\int \omega = \sum_{j \in \Gamma} \int \phi_j \omega,$$

provided the series converges.

If M is compact, then a partition of unity of M is a finite set. Hence, $\int \omega$ over a compacted manifold always exists. It is clear that the integral is independent of the chosen partition. As an application, we introduce the volume formula of a manifold (M, g) , where g is a parameterization of M . We define the volume differentiation of (M, g) by the k -form

$$dV = \sqrt{|G_g|} du,$$

where G_g is the Riemannian metric generated by g . Then the volume of M is

$$V(M) = \int_M 1.$$

The integral on an oriented manifold has the following properties.

Theorem A.1. Assume that M is k -dimensional oriented manifold. Let $-M$ denote the manifold having the opposite orientation with respect to M . Then we have the following.

1. For any continuous differential form ω of degree k on M ,

$$\int_{-M} \omega = - \int_M \omega.$$

2. For $a, b \in \mathbb{R}$,

$$\int_M a\omega_1 + b\omega_2 = a \int_M \omega_1 + b \int_M \omega_2.$$

3. Assume that at each coordinate system (W, u) , the k -form ω can be represented as $a \, du^1 \wedge \cdots \wedge du^k$, where a is continuous and $a \geq 0$. Then

$$\int_M \omega \geq 0,$$

where the equality holds if and only if $a = 0$.

4. If μ is a diffeomorphism from a k -dimensional oriented manifold M_2 to a k -dimensional oriented manifold M_1 , keeping the orientation, and ω is a continuous k -form on M_2 , then

$$\int_{M_1} \mu^* \omega = \int_{M_2} \omega.$$

To establish Stokes' Theorem, we introduce the concept of the manifold with boundary. Let the boundary of the closed half-space H^k (see (A.2)) be defined by the hyperplane

$$\partial H^k = \mathbb{R}^{k-1} \times 0 \subset \mathbb{R}^k.$$

In general, a manifold with boundary is defined as follows.

Definition A.6. A k -dimensional manifold $M \subset \mathbb{R}^n$ is called the one with boundary if for each $\mathbf{x} \in M$, there is a neighborhood W diffeomorphic to an open subset of H^k . The boundary of M , denoted by ∂M , is the set of the points mapped from the points in ∂H^k .

It is easy to see that ∂M is a $(k-1)$ -manifold. Note that each point $\mathbf{x} \in \partial M$ is also a point in M . Hence, the tangent space at \mathbf{x} , say, $T_{\mathbf{x}} M$, is a k -dimensional space with respect to M . However, since ∂M is $(k-1)$ -dimensional, its tangent space at \mathbf{x} , say, $T_{\mathbf{x}}(\partial M)$, is $(k-1)$ -dimensional space with respect to ∂M , and $T_{\mathbf{x}}(\partial M) \subset T_{\mathbf{x}} M$. Then $T_{\mathbf{x}}(\partial M)$ divides $T_{\mathbf{x}} M$ into two half-spaces H_+^k and H_-^k : all outward vectors are in H_+^k while all inward vectors in H_-^k . It can be proved that if M is oriented, then ∂M is oriented. The orientation of M induces an orientation of its boundary ∂M in the following way: We first select a positive oriented basis $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ of $T_{\mathbf{x}} M$ so that for $\mathbf{x} \in \partial M$ the vector \mathbf{v}_1 is an outward vector and the set $\{\mathbf{v}_2, \dots, \mathbf{v}_k\}$ is a basis of $T_{\mathbf{x}}(\partial M)$. Then the positive orientation of ∂M is determined by the basis $\{\mathbf{v}_2, \dots, \mathbf{v}_k\}$. This orientation on the boundary is called induced orientation. We now present the famous Stokes' Theorem.

Theorem A.2.(Stokes). Let M be a connected, oriented, and compact k -manifold with boundary and ω be a $(k-1)$ -form in C^1 . Then

$$\int_M d\omega = \int_{\partial M} \omega, \tag{A.5}$$

where the orientation over ∂M is induced from that over M .

A.3 Laplace-Beltrami Operator on Manifold

The Laplace operator, named after Pierre-Simon Laplace, operating on functions defined on Euclidean spaces, can be generalized to operate on functions defined on Riemannian manifolds. This more general operator goes by Laplace-Beltrami operator, after Laplace and Eugenio Beltrami. Like the Laplace operator, the Laplace-Beltrami operator is defined as the divergence of the gradient, and is a linear operator mapping functions to functions. The operator can be extended to operate on tensors as the divergence of the covariant derivative. Alternatively, the operator can be generalized to operate on differential forms using the divergence and exterior derivative. The resulting operator is called the Laplace-de Rham operator. Because in this book only Laplace-Beltrami operator is used, we introduce it in this section, skipping the discussion about the Laplace-de Rham operator.

Let (M, g) be a connected, oriented, and compact Riemannian k -manifold with the metric \mathbf{G}_g , and $\{(W, u)\}$ be a differentiable structure on M . Let the k -form having the expression $du^1 \wedge \cdots \wedge du^k$ in terms of the coordinates u be denoted by du . A function f defined on M is called square integrable if the k -form $|f|^2 dv$ is integrable. The space of all square integrable functions on M , denoted by $H(M)$, is a Hilbert space equipped with the inner product

$$\langle f, h \rangle = \int_M fh$$

The Laplace-Beltrami operator is a generalization of the Laplace operator on Euclidean space in (2.5). Recall that the Laplace operator on $C^2(\mathbb{R}^n)$ is defined as

$$\Delta f = \nabla \cdot \nabla f = \text{div grad } f,$$

where ‘‘div’’ denotes the divergent operator and ‘‘grad’’ denotes the gradient operator. We first generalize them on $C^2(M)$. Let $|\mathbf{G}|$ denote the determinant of the metric matrix $[g_{ij}]$: $|\mathbf{G}| = \det([g_{ij}])$. Let the coordinate mapping on M be denoted by h and the coordinates under h be $[u^1, \dots, u^k]$. Write $\partial_i = \frac{\partial}{\partial u^i}$. The divergence of a vector field $\mathbf{X} = [\mathbf{X}_1, \dots, \mathbf{X}_k]'$ on M is defined by

$$\text{div } \mathbf{X} = \frac{1}{\sqrt{|\mathbf{G}|}} \sum_{i=1}^k \partial_i \left(\sqrt{|\mathbf{G}|} \mathbf{X}_i \right).$$

The following Divergence Theorem can be derived from the Stokes’ Theorem.

Theorem A.3. Let M be a connected, oriented, and compact k -manifold with boundary, and $\mathbf{G} \stackrel{\text{def}}{=} \mathbf{G}_g$ is a metric on M . Let $\{(W, u)\}$ be a coordinate system on M . Let $\mathbf{X} = [\mathbf{X}_1, \dots, \mathbf{X}_k]^T$ be a C^1 vector field on M . Define the $(k-1)$ -form $\hat{d}^i u$ by

$$\hat{d}^i u = du^1 \wedge \cdots \wedge du^{i-1} \wedge du^{i+1} \wedge du^k.$$

Then the following equation holds.

$$\int_M \operatorname{div} \mathbf{X} = \int_{\partial M} \sum_{i=1}^k \sqrt{|\mathbf{G}|} \mathbf{X}_i d\hat{i} u.$$

The gradient of a scalar function f on M is a vector field defined by

$$\operatorname{grad} f = [(\operatorname{grad} f)^1, \dots, (\operatorname{grad} f)^k],$$

with

$$(\operatorname{grad} f)^i = \sum_{j=1}^k g^{ij} \partial_j f,$$

where $[g^{ij}]$ is the inverse matrix of $\mathbf{G}_g = [g_{ij}]$: $[g^{ij}][g_{ij}] = \mathbf{I}_n$.

Definition A.7. The Laplace-Beltrami operator on $C^2(M)$ is defined by

$$\Delta f = \operatorname{div} \operatorname{grad} f = \sum_{i=1}^k \sum_{j=1}^k \frac{1}{\sqrt{|\mathbf{G}|}} \partial_i \left(\sqrt{|\mathbf{G}|} g^{ij} \partial_j f \right). \quad (\text{A.6})$$

Assume that $f \in C^1$ is a compactly supported function on M and $\mathbf{X} \in C^1$ is an k -vector field on M . Applying the Divergence Theorem on the vector field $f\mathbf{X}$ and considering $f = 0$ on ∂M , we have

$$\int_M \operatorname{div} f \mathbf{X} = \int_{\partial M} \sum_{i=1}^k f \sqrt{|\mathbf{G}|} \mathbf{X}_i d\hat{i} u = 0.$$

Hence, we obtain

$$\int_M \langle \nabla f, \mathbf{X} \rangle = - \int_M f \operatorname{div} \mathbf{X}.$$

Particularly, choosing $\mathbf{X} = \operatorname{grad} h$ for a function $h \in C^2$ and applying (2.40), we get

$$- \int_M f \Delta h = \int_M \langle \nabla f, \nabla h \rangle, \quad (\text{A.7})$$

which yields

$$\int_M \|\nabla f\|^2 = - \int_M f \Delta f. \quad (\text{A.8})$$

From (A.7), we also obtain another useful formula

$$\int_M f \Delta h = \int_M h \Delta f. \quad (\text{A.9})$$

When the Riemannian metric \mathbf{G}_g is the identity so that $g_{ij} = g^{ij} = \delta_{ij}$. Then the formulas for divergent, gradient, and Laplace-Beltrami operators are reduced to the traditional expressions:

$$\operatorname{grad} f = \left[\frac{\partial}{\partial u^1}, \dots, \frac{\partial}{\partial u^k} \right],$$

$$\operatorname{div} \mathbf{X} = \sum_{i=1}^k \partial_i \mathbf{X}_i,$$

and

$$\Delta f = \sum_{i=1}^k \frac{\partial^2 f}{\partial(u^i)^2}.$$

Index

A

- Algorithm 300
d-rank approximation SVD algorithm, 301
ARPACK, 102
CMDS algorithm, 126
data_neighborhood, 58
Dijkstra's algorithm, 153, 158, 163
Dijkstra's algorithm, 19
Dmaps algorithm, 271
Dmaps algorithm of Graph-Laplacian type, 272
Dmaps of Laplace-Beltrami type, 274
eigenface, 109
EM-PCA algorithm, 102, 104
fast anisotropic transformation (FAT), 316
fast DR algorithms, 299
FAT, 320
FAT Isomaps, 322
Floyd's algorithm, 158
graph connection-checking algorithm, 84
Greedy anisotropic transformation (GAT), 318
greedy low-rank matrix approximation, 310
HLLE algorithm, 256
Implicitly Restarted Arnoldi Method, 102
interpolative RAT (IRAT), 319
Isomap algorithm, 156, 158, 186
L-ULV(A) rank-revealing, 305
L-ULV(L) rank-revealing, 304
landmark MVU, 194, 198
Leigs algorithm, 240, 241
LLE algorithm, 207, 208
low-rank approximation algorithm, 307
LTSA algorithm, 226
MVU algorithm, 186
PCA algorithm, 100, 101
PRAT of Type 1, 330
PRAT of Type 2, 330
PRAT of Type 3, 330
projection RAT (P-RAT), 320
random projection algorithm, 135
randomized algorithms, 311
randomized anisotropic transformation (RAT), 319
randomized greedy algorithm, 315
randomized low-rank matrix interpolation, 312
randomized SVD algorithm, 311, 314
robust graph connection algorithm, 85
RP face projection, 145
SDP, 186
SDP software package—CSDP, 187
SDP software package—DSDP, 187
SDP software package—PENSDP, 187
SDP software package—SBmeth, 187
SDP software package—SDPA, 187
SDP software package—SDPLR, 187
SDP software package—SDPT3, 187, 188
SDP software package—SeDumi, 187

self-tuning Dmaps algorithm, 276
semidefinite programming, 181
semidefinitiy programming, 186
SPCA algorithm, 104
square_distance, 56, 57
UTL rank-revealing, 304
UTV Expansion Pack, 306
UTV Tools, 306
VSV rank-revealing, 306

Application
2-dimensional visualization, 192
classification, 17
data feature extraction, 2
data visualization, 2
eigenfaces, 108
face recognition, 2, 3, 109, 145
fingerprint identification, 2
hyperspectral image analysis, 2, 112
keyword search, 6
LLE in image ordering , 214
pattern recognition, 17
RP in document sorting, 146
sensor localization, 190, 192
Supervised LLE, 215
text document classification, 2
visual perception, 172
visualization, 18

B

Barycentric coordinates
Hermite mean value interpolation, 205
Lagrange interpolating property, 204
linearity, 204
positivity, 204

C

Curse of the dimensionality, 8
concentration of distances, 11
concentration of norms , 11
diagonals of cube, 11
empty space phenomenon, 9
tail probability, 10
volume of a thin spherical shell, 9
volume of cubes and spheres, 9

D

Data, 51
 ε -neighborhood, 52
B-coordinates, 56
k-nearest neighbors, 19
k-neighborhood, 52
barycentric coordinates, 204
compressive sensing, 22
configuration, 79, 82, 115, 117, 125
configurative points, 117
constraints on DR output, 82
data compression (DC), 22
data geometry, 51
data graphs, 60
data matrix, 30
data models in DR, 79
dataset, 3
density of data, 292
dissimilarities, 79, 115, 116
dissimilarity, 51, 79, 119, 152
exact configuration, 123
extrinsic dimension, 52, 79
feasible set, 186
geometric structure, 18
graph, 152, 156
graph neighborhood, 83
high-dimensional data, 1, 51
HSI data, 152
intrinsic configuration dimension, 123
intrinsic dimension, 79
landmarks, 194
lossless compression, 22
lossy compression, 22
manifold coordinate representation, 222
manifold neighborhood, 83
multiscale diffusion features, 292
multiscale diffusion geometry, 290
multiscale geometric structures, 267
nearest neighbors, 12
neighborhood, 51, 52, 90, 152
neighborhood system, 18, 60
noise, 80
noise estimation, 81
object vectors, 79
observed data, 18
olivetti faces, 3
point, 3

- point cloud, 3
- similarities, 79, 115, 116
- similarity, 18, 51, 79, 152
- source coding, 22
- tangent coordinate representation, 222
- tangent coordinates, 251
- text documents, 6
- training set, 108
- vector, 3
- visible data, 18
- Data graph
 - connection consistence, 83
 - consistent data graph, 83
 - graph consistent conditions, 83
 - local dimension consistence, 83
- Differential form, 339
 - inverse image, 340
 - wedge product, 340
 - degree, 339
- Digraph
 - adjacency matrix, 64
 - arc, 64
 - arrow, 64
 - boundary in-volume of cluster, 66
 - boundary out-volume of cluster, 66
 - direct successor, 64
 - directed edge, 64
 - directed path, 65
 - head of arc, 64
 - in-degree, 64
 - in-neighborhood, 65
 - in-volume of cluster, 66
 - in-volume of node, 66
 - inverted arc, 64
 - inverted path, 65
 - isomorphism, 64
 - label graph, 64
 - linear digraph, 65
 - oriented graph, 64
 - out-degree, 64
 - out-neighborhood, 65
 - out-volume of cluster, 66
 - out-volume of node, 66
 - predecessor, 64
 - predecessor set, 65
 - regular digraph, 65
 - regular digraph of in-degree k , 65
 - regular digraph of out-degree k , 65
 - strongly connected digraph, 65
 - subgraph, 65
- successor set, 65
- symmetric digraph, 65
- symmetric pair of nodes, 64
- tail of arc, 64
- weakly connected digraph, 65
- weight matrix, 65
- Dimension
 - extrinsic dimensions, 12, 17
 - intrinsic configuration
 - dimension, 82, 83
 - intrinsic dimensions, 12
 - intrinsic dimensions estimation, 12
- Lebesgue covering dimension, 14
- linearly intrinsic dimension, 13
- manifold dimension, 14
- topological dimension, 14
- Dimensions, 12
- Distance
 - l_∞ distance, 53
 - angular distance, 53
 - derivative distance, 55
 - diffusion distance, 267
 - directionally weighted distance, 55
 - distance in homogeneous Sobolev spaces, 55
 - edge distance, 55
 - Euclidean distance, 119, 152
 - Fourier distance, 54
 - Fourier phase distance, 54
 - Fourier phase-difference distance, 55
 - geodesic distance, 19, 46, 152, 153, 175
 - graph distance, 152, 153, 175
 - Mahalanobis distance, 56
 - Manhattan distance, 118
 - path distance, 153
 - S-distance, 175
 - template distance, 53
 - wavelet distance, 54
- Dmaps, 267
 - graph-Laplacina type, 269
 - Laplace-Beltrami type, 269
 - self-tuning Dmaps, 276
- DR, 1, 3, 12
 - input data of the first type, 79
 - centralization constraint on output, 83
 - classical multidimensional scaling (CMDS), 123
 - classical multidimensional scaling (LMDS), 115

CMDS, 126, 191
 diffusion maps, 267
 generalized Leigs, 240
 geometric approach, 17, 18
 geometric embedding, 18
 hard dimensionality reduction, 17
 Hessian locally linear embedding (HLLE), 249
 input data of the second type, 81
 Isomaps, 151, 181
 kernel, 18
 Landmark MVU, 181
 Laplacian eigenmaps, 235, 249
 linear data model, 79
 Linear method, 18
 LMVU, 194
 local tangent space alignment (LTSA), 221
 locally linear embedding (LLE), 203, 206
 maximum variance unfolding (MVU), 181
 MDS, 181
 multidimensional scaling (MDS), 115
 multivariate analysis, 18
 MVU, 181
 nonlinear data model, 80
 nonlinear method, 18
 orthogonality constraint on output, 83
 output data, 82
 PCA, 126, 299
 principal component analysis (PCA), 95
 random projection (RP), 131
 scaled output, 83
 semidefinite embedding (SDE), 181
 SLLE, 215
 soft dimensionality reduction, 18
 spectral method, 18
 t-SLLE, 215

E

Embedding
 linear embedding, 30
 orthogonal embedding, 13, 30

F

Face recognition

eigenfaces, 109
 face space, 108
 illumination variations, 111
 large-scale features, 111
 small-scale features, 111
Function
 coordinate function, 32
 feature function, 267
 Gamma function, 33
 homogeneous linear function, 32
 linearly independent, 32
 local tangent coordinate functions, 256
 Taylor's formula, 33
Functional
 Hessian functional, 250
 Hessian functional in manifold coordinates, 252
 Hessian functional in tangent coordinates, 252
 linear functional, 31, 32
 local tangent functional, 256
 manifold Hessian functional, 250
 tangent Hessian functional, 252

G

Graph, 51
k-connected graph, 62
k-edge-connected graph, 62
k-regular graph, 61, 67
k-vertex-connected graph, 62
 adjacency matrix, 62, 73
 adjacent nodes, 61
 boundary volume of cluster, 63
 cluster, 63
 complete graph, 61, 70, 115
 connected component, 62, 70
 contracting operator, 75
 contraction, 75
 degree of vertex, 61
 diameter, 72
 digraph, 60, 64
 directed graph, 60, 64
 Dirichlet sum, 69
 disconnected graph, 63
 edge set, 156
 edgeless graph, 60
 edges, 60
 end of edge, 60
 end vertex of edge, 60
 endpoint of edge, 60
 equivalent, 62

extended neighborhood of node, 61
graph distance, 72
graph refinement, 185
harmonic eigenfunction, 69, 73
homomorphic graphs, 62
homomorphism, 62
isolated point, 67
isolated vertex, 70, 73
isomorphism, 62
label graph, 62
Laplacian, 67
Laplacian matrix, 73
Laplacian on weighted graph, 74
locally complete graph, 185
loop, 61
neighborhood of node, 61
node degree, 63
node volume, 63
nodes, 60
non-normalized Laplacian matrix, 67
path, 62
points, 60
Rayleigh quotient, 69
regular graph, 61
self edge, 61
simple digraph, 64
simple graph, 60
spectral analysis, 67
spectral graph theory, 18, 51, 67
spectrum, 68
subgraph, 61
trivial graph, 60
undirected graph, 60
unweighted graph, 63
vertex, 60
vertical set, 156
volume, 72
volume of vertex, 74
weight, 62
weight matrix, 18, 62, 206
weighted graph, 18, 62
weighted Laplacian matrix, 74

H

Heat diffusion
heat capacity, 292
mass density, 292
thermal conductivity, 292
thermal diffusivity, 292
High-dimensional data
facial images, 3

facial databases, 3
fingerprint, 1
fingerprints, 1
hand-writing letters and digits, 1
handwriting letters and digits, 5
hyperspectral images, 1
hyperspectral images (HSI), 6
images, 1
speech signals, 1
text documents, 1
videos, 1
Hyperspectral image
band-image, 113
hyperspectral signature, 112
raw data, 52
spectral curve, 112
virtual band-image, 113

I

Image, 52
eigenface, 4
gray-level, 52
intensity, 52
Intrinsic dimension estimating method
capacity dimension, 16
correlation dimension, 15
fractal dimension, 15
geometric approach, 14
multiscale estimation, 16
nearest neighbor distance, 15
projection technique, 14
Isomaps
 ε -Isomap, 156
 k -Isomap, 156
constant-shift technique, 156

K

Kernel
constant-shifted Isomap kernel, 154
diffusion kernel, 291
Dmap kernel, 269
Dmaps kernel of graph-Laplacian type, 270
Dmaps kernel of Laplace-Beltrami type, 270
Gaussian kernel, 35, 237
Gaussian-type diffusion kernel, 267

graph Laplacian, 239
 HLLE kernel, 251, 252, 255
 Isomap kernel, 154, 158
 Leigs kernel, 239, 269
 LLE kernel, 208
 LTSA kernel, 227
 non-normalized diffusion kernel, 288
 Normalization of Dmaps kernels, 269
 PCA kernel, 100
 sparse, 20

L

Linear DR method
 classical multidimensional scaling (CMDS), 18
 principal component analysis (PCA), 13, 18
 random projection, 18
LLE
 invariance constraint, 206
 sparseness constraint, 206
LTSA
 global alignment, 225
 local coordinate representation, 224

M

Machine learning, 55, 107
 examples, 107
 feature extractor, 107
 feature function, 107
 feature vector, 55
 manifold learning, 29
 semi-supervised learning, 191
 supervised learning, 107, 191
 unsupervised learning, 107, 191

Manifold, 18
 k -manifold, 36
 arc length of curve, 46
 atlas, 38
 compact manifold, 175
 connected manifold, 175
 convex manifold, 175
 coordinate neighborhood, 36
 curvature, 16
 developable surface, 22
 differentiable structure, 38, 42
 differential manifold, 36
 discrete form, 51

embedded, 13
 global theory, 29
 hyperplane, 13, 31, 38, 79
 linear, 13
 linear manifold, 29
 local theory, 29
 Möbius strip, 342
 manifold geometry, 29
 manifold with boundary, 344
 minimum branch separation, 177
 minimum radius of curvature, 177
 neighborhood, 14
 open covering, 14
 orientation, 342
 parameterization, 12, 36
 parametrization, 29
 Riemannian manifold, 43
 S-curve, 22
 sampling data set, 51
 simple manifold, 38
 spherical surface, 342
 submanifold, 38
 Swiss roll, 22
 tangent place, 38
 tangent space, 20, 29, 40, 222
 tangent vector field, 43
 volume, 343
 coordinate system, 36
 Half-space, 342
 induced orientation, 344
 integral, 341
 oriented manifold, 344
Mapping
 coordinate mapping, 36
 diffeomorphism, 36
 differentiable homeomorphism, 36
 diffusion maps, 290
 feature mapping, 290
 homeomorphism, 36
 JL-embeddings, 133
 Lipschitz embedding, 131
 Lipschitz mappings, 131
 random projection, 133
 smooth, 36
Matrix, 30
 array of distance matrices, 82
 best low-rank approximation, 300
 best rank-revealing approximation, 300
 centered data matrix, 121

centered Gram matrix, 185
 centering Gram matrix, 119,
 122, 124
 centering symmetric matrix, 122
 centralizing matrix, 121
 Cholesky decomposition, 120
 coordinate-change matrix, 42
 covariance matrix, 98
 data matrix, 119, 120
 diagonal matrix, 30
 Euclidean distance matrix, 115,
 119, 120
 Euclidean square-distance matrix,
 119, 124
 general inverse, 31
 Gram matrix, 115, 119, 120
 greedy low-rank approximation,
 309
 Hadamard product, 127, 254
 Hadamard square, 309
 Hessian matrix, 33
 low-rank approximation, 299
 manifold Hessian matrix, 250
 matrix decomposition, 299
 matrix with orthonormal columns,
 30
 matrix with orthonormal rows,
 30
 Nyström approximation, 307
 orthogonal matrix, 30
 positive definite matrix, 30
 positive semi-definite (psd) ma-
 trix, 120
 positive semi-definite matrix, 30
 QR decomposition, 31
 random matrix, 131
 random matrix of Type-1, 134
 random matrix of Type-2, 134
 random matrix of Type-3, 134
 rank, 30
 rank-revealing factorization, 299
 rank-revealing LU factorization,
 302
 rank-revealing QR factorization,
 302
 rank-revealing ULV factorization,
 303
 rank-revealing URV factorization,
 303
 rank-revealing UTV factorization,
 302
 Schur decomposition, 302
 shifted Gram matrix, 121
 similarity matrix, 81
 SVD decomposition, 299

tangent Hessian matrix, 252
 transpose, 30
MDS
 classical MDS (CMDs), 118, 123
 classical multidimensional scal-
 ing, 123
 metric MDS, 117
 metric multidimensional scaling
 (MMDS), 123
 nonmetric MDS, 118
 replicated MDS, 118
 unweighted MDS, 118
 weighted MDS, 118

Metric
 canonic metric, 45
 Euclidean metric, 45, 118, 119,
 152
 geodesic metric, 152, 156
 graph metric, 156
 Minkowski metric, 118
 Riemannian metric, 43

N

Nonlinear DR kernel, 90
 distance approach, 88
 weight approach, 89
Nonlinear DR method
 autoencoders, 21
 diffusion maps (Dmaps), 20
 generative topographic map, 21
 Hessian locally linear embedding
 (HLLE), 20
 Isomaps, 19
 Kohonen map, 21
 Laplacian eigenmaps (Leigs), 20
 local multidimensional scaling,
 19
 local tangent space alignment
 (LTSA), 20
 locally linear embedding (LLE),
 19
 machine learning, 21
 manifold sculpting, 19
 maximum variance unfolding
 (MVU), 19
 neural network, 21
 NeuroScale, 21
 self-organizing feature map
 (SOFM), 21
 self-organizing map (SOM), 21
 semidefinite embedding (SDE),
 19

O

Operator

- derivatives, 39
- diffusion operator, 268
- dilation, 31
- discrete Laplacian, 238
- discrete Laplace-Beltrami, 237
- discrete local Hessian, 255
- divergent, 49, 346
- exponential formula, 236
- feature extractor, 292
- gradient, 33, 39, 49, 346
- Hessian, 20, 249, 250, 252
- Laplace-Beltrami, 20, 49, 70, 235, 236, 268
- Laplacian, 34, 67
- Neumann heat diffusion, 20, 35, 236, 268
- Nyström approximation, 307
- positive and self-adjoint, 236
- projection, 30
- rotation, 31
- self-joint operator, 68
- semi-group, 290
- shift, 31

P

PCA

- principal components, 96, 99, 108
- principal directions, 96, 99, 108

PDE

- Bobin boundary condition, 34
- Cauchy initial problem, 35
- diffusion kernel, 35
- Dirichlet boundary condition, 34
- exponential formula, 35
- Green's function, 35
- heat diffusion equation, 35
- Laplacian eigenvalue problem, 34
- Neumann boundary condition, 34

R

- Random vector, 99, 100
 direction of maximum variance,
 99

- first principal component, 99
- first principal direction, 99
- principal components, 99
- principal directions, 99
- random component, 99

S

Space

- affine space, 186
- dual, 31
- feature space, 290
- Hilbert space, 32
- Hausdorff space, 36
- Lebesgue space, 33
- metric space, 16

T

Theorem

- Courant-Fischer Theorem, 69
- Divergence Theorem, 48, 345
- Inverse function theorem, 39
- Johnson-Lindenstrauss Lemma, 133
- Young and Householder Theorem, 123
- Stokes' Theorem, 344

U

Undirected graph

- connected graph, 62, 65
- connected vertices, 62
- disconnected graph, 62
- disconnected vertices, 62
- isomorphic graphs, 62
- linear graph, 61

W

- Wavelet, 54
 discrete wavelet transform, 54
 DWT, 54
 multi-level DWT, 54