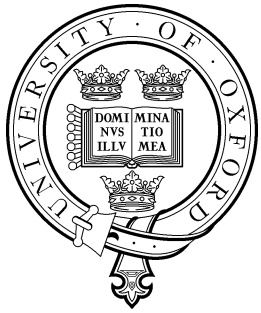
Diffusion Maps: Analysis and Applications



Bubacarr Bah
Wolfson College
University of Oxford

A dissertation submitted for the degree of
MSc in Mathematical Modelling and Scientific Computing

September 2008

This thesis is dedicated to
my family, my friends and my country
for the unflinching moral support and encouragement.

Acknowledgements

I wish to express my deepest gratitude to my supervisor Dr Radek Erban for his constant support and valuable supervision. I consider myself very lucky to be supervised by a person of such a wonderful personality and enthusiasm.

Special thanks to Prof I. Kevrekidis for showing interest in my work and for motivating me to do Section 4.3 and also for sending me some material on PCA.

I am grateful to the Commonwealth Scholarship Council for funding this MSc and the Scholarship Advisory Board (Gambia) for nominating me for the Commonwealth Scholarship award. Special thanks to the President Alh. Yahya A.J.J. Jammeh, the SOS and the Permanent Secretary of the DOSPSE for being very instrumental in my nomination for this award.

My gratitude also goes to my employer and former university, University of The Gambia, for giving me study leave.

I appreciate the understanding and support given to me by my wife during these trying days of my studies and research.

Contents

List of Figures

ABSTRACT

Diffusion Maps :

analysis and applications

Bubacarr Bah

2008

A lot of the data faced in science and engineering is not as complicated as it seems. There is the possibility of finding low dimensional descriptions of this usually high dimensional data. One of the ways of achieving this is with the use of diffusion maps. Diffusion maps represent the dataset by a weighted graph in which points correspond to vertices and edges are weighted. The spectral properties of the graph Laplacian are then used to map the high dimensional data into a lower dimensional representation.

The algorithm is introduced on simple test examples for which the low dimensional description is known. Justification of the algorithm is given by showing its equivalence to a suitable minimisation problem and to random walks on graphs. The description of random walks in terms of partial differential equations is discussed. The heat equation for a probability density function is derived and used to further analyse the algorithm.

Applications of diffusion maps are presented at the end of this dissertation. The first application is clustering of data (i.e. partitioning of a data set into subsets so that the data points in each subset have similar characteristics). An approach based on diffusion maps (spectral clustering) is compared to the K-means clustering algorithm. We then discuss techniques for colour image quantization (reduction of distinct colours in an image). Finally, the diffusion maps are used to discover low dimensional description of high dimensional sets of images.

Chapter 1

INTRODUCTION

1.1 The dimensionality reduction problem

Today's information age inundates us with lots of complex and high dimensional data that we have to simplify in order to better comprehend it or for other reasons. In many fields like machine learning, statistical data analysis and bio-informatics we try to represent high dimensional data sets involving a large number of variables by a low dimensional description with a smaller number of free parameters. Apparently, we can represent such high dimensional data spaces with a few coordinates without loosing the substances of interest in the data. The process of discovering this low-dimensional description of a high dimensional system is the main theme of dimensionality reduction.

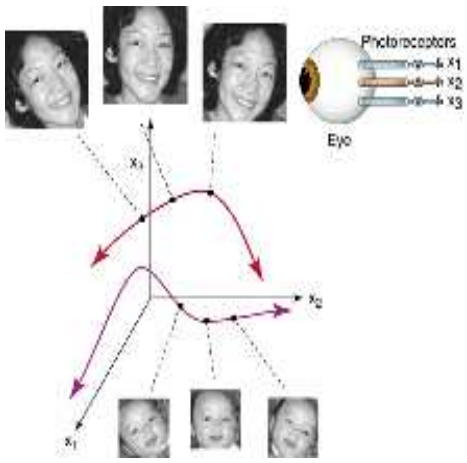


Figure 1.1: From Manifolds in visual perception from H. Sabastian Seung, SCIENCE 290:2268-2269 (22 December 2000). Reprinted will permission from AAAS. [28]

For example, in Figure 1.1 we have a pair of grayscale images of rotating faces shot by a fixed camera. Each pixel of the images represents a brightness value. If each image

is composed of n by n pixels, then each image is a data point in $\mathbb{R}^{n^2}$. As the faces are rotated they trace out non-linear curves embedded in the image space $\mathbb{R}^{n^2}$. Thus the images can be represented by a single parameter which is the angle of rotation which means the images are intrinsically one dimensional. Even if in addition to brightness, we consider scaling, illumination and other sources of continuous variability, the images would still have an intrinsic dimensionality lower than $\mathbb{R}^{n^2}$ [1, 23].

Furthermore, in the natural science for instance, data is usually sampled from a complex biological, chemical or physical dynamical system. Most of these system have multiple and clearly separated time scales. For example in interacting particle systems slow time scales depict the evolution of mean density profiles, while the fast time scales depict density fluctuations about the mean density profiles. Typically, these coarse time evolution of the high dimensional system can be represented by a few dynamically relevant variables. Therefore, people working in these fields are faced with dimensionality reduction to these coarser scales [19].

Several techniques have been developed to solve the dimensionality reduction problem or in other words, the problem of finding meaningful structures and geometric descriptions of high dimensional data. These techniques can be generally categorised into linear and nonlinear methods. Linear methods include classical methods like principal component analysis (PCA) and multidimensional scaling (MDS). The nonlinear methods include kernel PCA, the so-called "kernel eigenmaps" and diffusion maps which is the main focus of this work.

In the rest of the introduction I give a brief description of the diffusion map algorithm and present two illustrative examples. In the first example I recover a line from a 3D data space sampled with some noise. In the second, I recover a nonlinear curve embedded in a 3D input space. In both, besides diffusion maps I used PCA for the same task in a bid to compare results of the two methods.

The dissertation's outline is as follows. Chapter 1 is an introduction, presenting the algorithm and illustrative examples ending with some general discussion. The justification of the algorithm is given in Chapters 2 and 3. Chapter 2 justifies the choice of weights while chapter 3 interprets the algorithm as some form of random walk on graphs. Since the algorithm is a sparse matrix problem, chapter 4 discusses solution of sparse matrix problems. Chapter 5 shows applications on clustering confirming the connection between diffusion maps and clustering, and an application on recovering low dimensional description of a set of images. These chapters a followed by a conclusion and appendices.

1.1.1 Algorithm

Basically, all what dimensionality reduction algorithms do is represent a given data set $\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)}\} \in \mathbb{R}^n$ where $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})$ and $i = 1, 2, \ldots, k$ by $\mathbf{Y} = \{\mathbf{y}^{(1)},\ \mathbf{y}^{(2)}, \ldots,\ \mathbf{y}^{(k)}\} \in \mathbb{R}^m$ where $\mathbf{y}^{(j)} = (y_1^{(j)}, y_2^{(j)}, \ldots, y_m^{(j)})$ and $j = 1, 2, \ldots, k$ such that $m \ll n$.

Consider $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)} \in \mathcal{M}$ where $\mathcal{M}$ is the manifold embedded in $\mathbb{R}^n$. A manifold is an abstract mathematical space with a complicated global structure in which every point has a neighbourhood which resembles a Euclidean space. Now, given these k points $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(k)}$, we model the data by a weighted graph with data points as *nodes* and weighted *edges* connecting nodes. A matrix representation of a graph is known as the Laplacian matrix. The eigenvectors of the Laplacian matrix can be viewed as coordinates of the data set and hence determine the embedding map [1]. The procedure is outlined below in four key steps.

1. Construct the adjacency matrix, $\mathbf{W}$, of the graph. The entries of $\mathbf{W}$ are the weights along the edge connecting corresponding nodes which for any given nodes i and j is determined by (the heat kernel):

$$W_{ij} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\varepsilon}\right). \tag{1.1}$$

   where $\|.\|$ is the Euclidean norm, ε is an appropriately chosen parameter discussed in Section 2.2 of Chapter 2. Note that $\mathbf{W} \in \mathbb{R}^{k\times k}$ is symmetric.

2. Construct the diagonal $k \times k$ normalisation matrix $\mathbf{D}$ and the Laplacian matrix $\mathbf{L}$, both respectively given by:

$$D_{ii} = \sum_{j=1}^{k} W_{ji} \qquad \text{and} \qquad \mathbf{L} = \mathbf{D} - \mathbf{W}. \tag{1.2}$$

   Note also that $\mathbf{L}$ is symmetric and positive semi-definite and since $\mathbf{W}$ is symmetric the diagonal entries of $\mathbf{D}$ are either the row or column sums of $\mathbf{W}$.

3. Compute eigenvalues and eigenvectors of the generalised eigenvalue problem

$$\mathbf{L}\mathbf{y} = \lambda\mathbf{D}\mathbf{y} \tag{1.3}$$

   where $\mathbf{y} \in \mathbb{R}^k$ are the column vectors of $\mathbf{Y}$.

   For ordered eigenvalues $0 = \lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_{k-1}$ the corresponding eigenvectors $\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_{k-1}$ are solutions to (1.3).

4. Drop $\mathbf{y}_0$ because it corresponds to $\lambda_0 = 0$, and use the next m eigenvectors to represent the $n-$dimensional input space by the following *diffusion map*:

$$\mathbf{\Psi}_m : \mathbf{x}^{(i)} \to (y_1^{(i)}, \ldots, y_m^{(i)}) \tag{1.4}$$

where $\mathbf{\Psi}_m : \mathbb{R}^n \to \mathbb{R}^m$ hence $\mathbf{\Psi}_m : \mathbf{X} \to \mathbf{Y}$ as required [1, 8].

1.2 Illustrative Examples

To demonstrate that the above algorithm works, here are two illustrative examples. In these examples I generated the data in 3D with some additive noise. The underlying low dimensional representation is known and it is 1D in both cases. The first one is a line and the second is a curve. The diffusion map algorithm recovered these underlying 1D structures in the higher dimensional (3D in this case) input space. Thus the algorithm discovered the true ("good") coordinate or parameter.

A comparison of diffusion maps to PCA is also done along side. Showing how the diffusion map algorithm out performs PCA when it comes to nonlinear data. A brief description of the PCA technique is given in Appendix 1 (for a detailed one see [24]) and for a quick implementation of the diffusion map algorithm in Matlab a pseudo code is given in Appendix 2.

1.2.1 A linear example

To get the data set, I generated 1000 points, t_i where $i = 1, \ldots, 1000$, which are uniformly distributed random numbers in $[0, 10]$ and then I used the relation (1.5):

$$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \\ -3.2 \end{pmatrix} t_i + \begin{pmatrix} 0 \\ 7 \\ 1 \end{pmatrix} + \text{noise} \tag{1.5}$$

where the noise is normally distributed with zero mean and unit variance.

These points are plotted in Figure 1.2. The same data is plotted in the two graphs. The right plot is colour coded according to the index i which is depicted by the colour bar next to the figure. This shows that the points were not ordered before the application of the diffusion map.

Then using diffusion map algorithm, I found the six most important eigenvectors of (1.3). That is the six eigenvectors corresponding to the six smallest eigenvalues of (1.3). Similarly, using a PCA algorithm I generated six largest eigenvectors corresponding to the six principal components of the data. With a scatter diagram, I

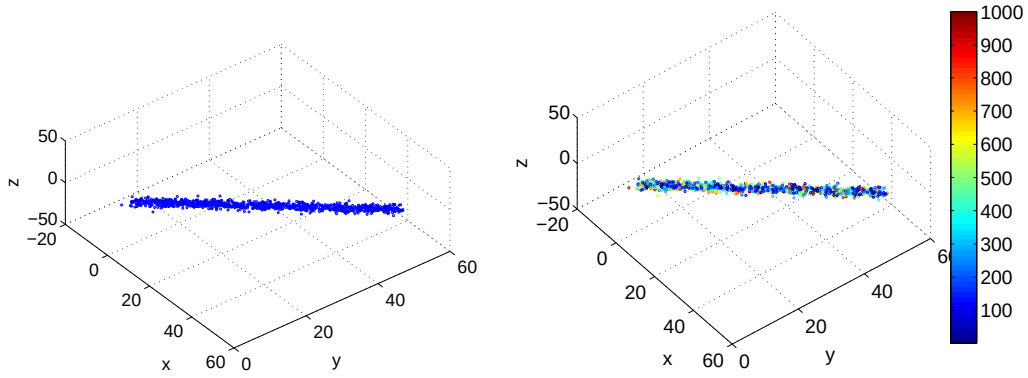


Figure 1.2: A plot of the same randomly generated data points in 3D with the right plot colour coded.

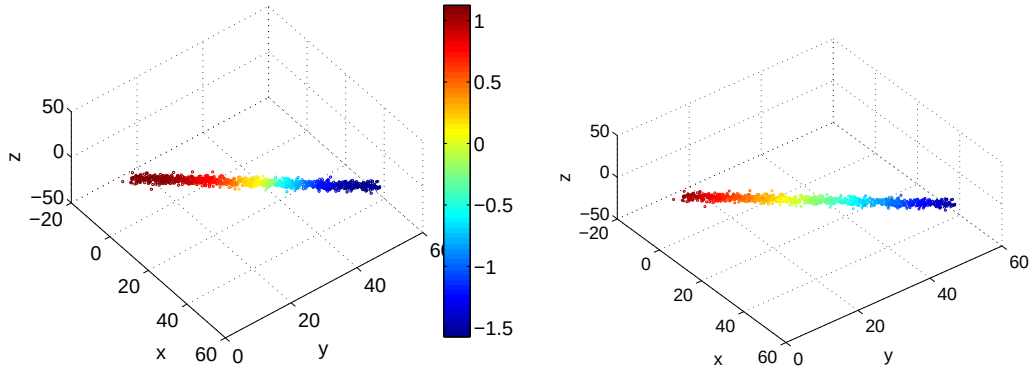


Figure 1.3: Approximation of the input data using the 2nd eigenvector of diffusion map (left) and the 1st principal component of PCA (right).

coloured the points according to the 2nd eigenvector of the diffusion map and the result is in the left panel of Figure 1.3. Similarly, the right panel of Figure 1.3 shows a plot of the same points coloured according to the 1st eigenvector of the PCA.

Again points with same colouring occupy the same position from some reference point along the line. Points coloured blue are on one extreme end while points coloured red on the other extreme end, see colour bar beside left plot. The colour variation traces a line, thus showing the 1D embedding of the seemingly high (three) dimensional input space. There is a clear agreement between the two plots. So, the two algorithms must be doing the same thing here. It is convincing though, to see that diffusion maps could recover principal components like the famous and trusted PCA does.

It could also be said that the colour variation depicts the monotonic increase of t on a scale from 0 to 10. Thus the methods (diffusion maps and PCA) clearly recovers

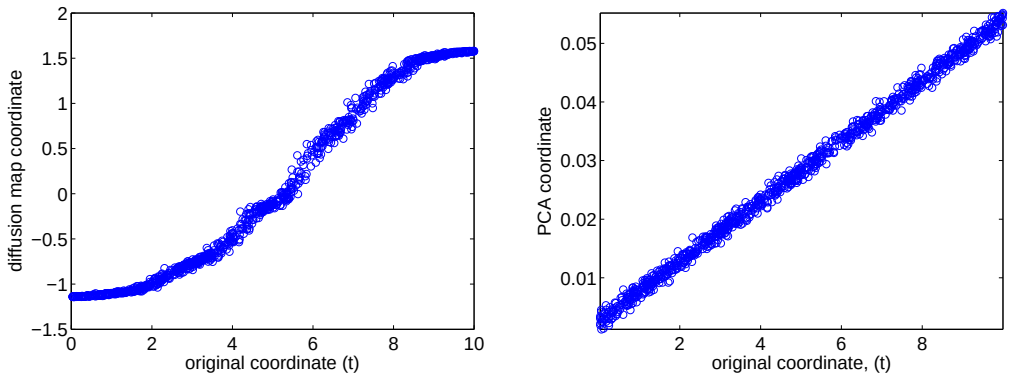


Figure 1.4: A map of the original variable t with the approximate variables from (left) the diffusion map and (right) the PCA.

the "good" coordinate which is t. This is further confirmed in the plots of Figure 1.4, where I plotted the original 'good 'coordinate used to generate the points, with the approximate 'machine'coordinate recovered by the diffusion maps and PCA. The diffusion map coordinate (the second eigenvector) against t in the left panel and t against the first principal component of PCA in the right plot. Both plots show a one-to-one map. This agrees with earlier findings of [28, 21].

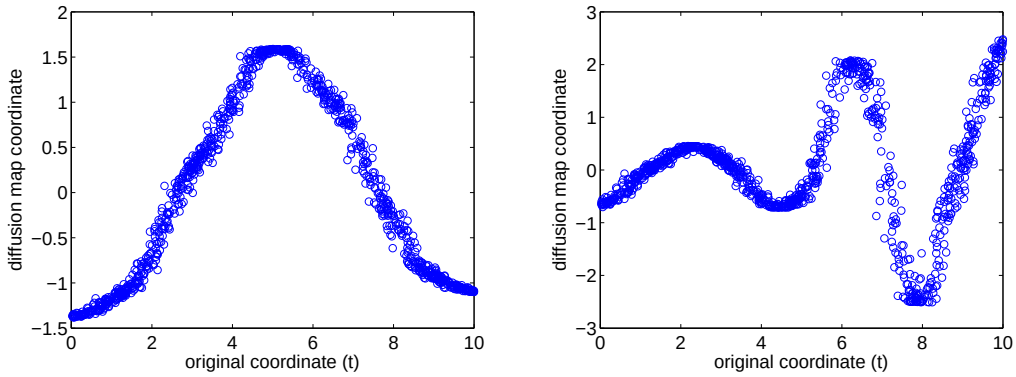


Figure 1.5: A map of the original variable t against the approximate variables: (left) 3^{rd} and (right) 6^{th} eigenvectors of the diffusion map.

A further plot of the 3^{rd} and 6^{th} eigenvectors of the diffusion map against t in Figure 1.5 shows the mapping no longer becomes one-to-one. So the best approximation of the good coordinate for diffusion map is the 2^{nd} eigenvector.

1.2.2 A nonlinear example

The data for a nonlinear manifold was generated in a similar way to the linear example above. Using arc length as the nonlinear parameter, I generated the data set as follows:

$$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = (1 + a_i) \begin{pmatrix} \sin a_i \\ \cos a_i \\ \sin 2a_i \end{pmatrix} + 0.1 \times \text{noise} \tag{1.6}$$

where $a_i = 10(1 - r_i)^{1.5}$, $i = 1, \ldots, 1000$, and r_i is a set of uniformly distributed random numbers on the unit interval while the noise is normally distributed with zero mean and unit variance.

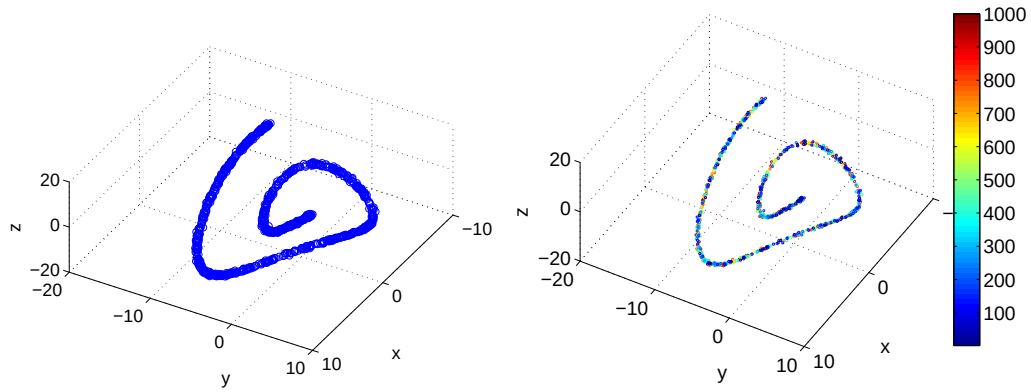


Figure 1.6: A plot of the same randomly generated data points along a nonlinear manifold in 3D with right plot colour coded.

As before too, the data set is plotted in Figure 1.6 with the right plot colour coded according to the index i to show the disorder of the points in the curve. In Figure 1.7 the second eigenvector of the diffusion map is used to colour the left scatter diagram while the first eigenvector of the PCA is used to colour the right scatter diagram.

Again we see that the diffusion maps were able to recover the embedded low-dimensional manifold, going by the colour variation of the left plot of Figure 1.7. This time the PCA produced a different result from the diffusion map. We can see that the colour code in the right plot of Figure 1.7 does not show any consistent ordering. Thus it fails to recover the 1D curve recovered by the diffusion map. This is not surprising since what PCA basically does is try to fit a "linear plane" through the data.

Figure 1.8 is a plot of the true coordinate, arc length versus the approximated coordinates which are the 1^{st} eigenvector of the PCA and 2^{nd} eigenvector of the diffusion map respectively. The diffusion map gives a one-to-one map while the PCA

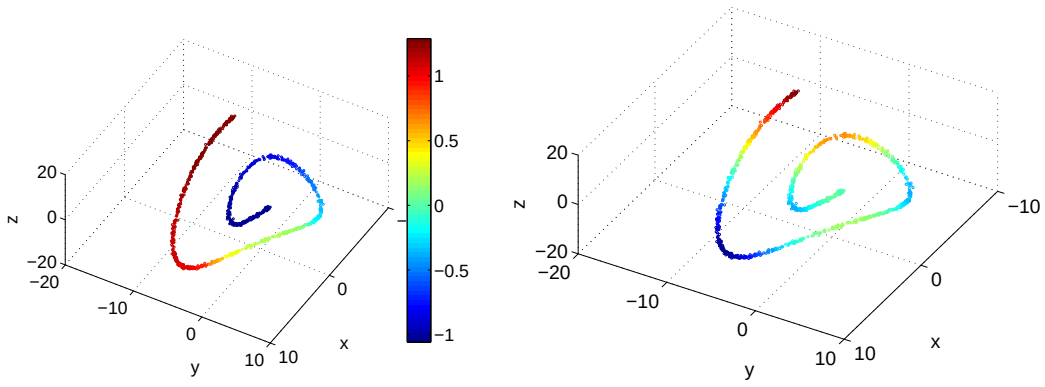


Figure 1.7: A plot of the data set coloured using the 2nd eigenvector of the diffusion maps (left) and the 1st principal component of PCA (right).

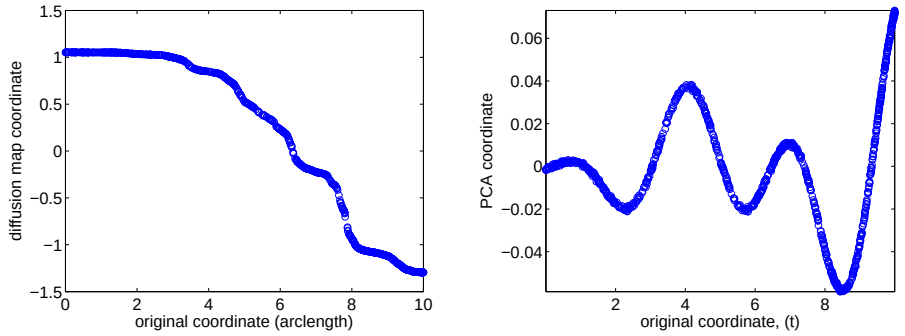


Figure 1.8: A map of the original variable *arclength* with the approximate variables from (left) the diffusion map and (right) the PCA.

is no longer one-to-one. Therefore diffusion maps are a better dimensional reduction technique than PCA when the data is nonlinear.

1.3 Discussion

In this section, I would comment on the algorithm and the examples presented above, give a brief survey of dimensionality reduction techniques and outline the key features of the diffusion map technique which motivated me to study or work on it.

Firstly, using the heat kernel (1.1) for the choice of weights is geometrically motivated since intuitively it is locality preserving. However, the choice of ε in (1.1) is not entirely trivial and it is discussed in Chapter 2. Furthermore, there are other ways of connecting points and choosing weights. For instance, we could connect points using the $s-$nearest neighbours criterion, by which nodes i and j are connected if i is among the s nearest neighbours of j or vice versa, where $s \in \mathbb{N}$. Then we choose $W_{ij} = 1$ or 0

if the nodes i and j are connected or otherwise respectively. This circumvents any difficulty of choosing ε, but it is less geometrically intuitive [13].

Secondly, the above illustrative examples show that the diffusion map algorithm works. This is confirmed by the pictorial representation of the results. It was possible to display the results by way of plotting because the space is 3D. However, for high dimensions this is not possible. It is good to know that even though we may not be able to display its outcome diffusion can recover low dimensional representation of high dimensional spaces.

As mentioned earlier, a plethora of dimensionality reduction methods exist. Classical linear methods are PCAs and MDS. Nonlinear methods include kernel PCA, Isometric Feature Mapping (ISOMAP) proposed by Tenenbaum et al. in 2000 [28] and kernel eigenmap methods like locally linear embedding (LLE) of Roweis and Saul (2000) [21], Laplacian eigenmaps by Belkin and Niyogi (2001) [1], Hessian eigenmaps (Hessian LLE) proposed by Donoho and Grimes (2003) [6] and local tangent space alignment (LTSA) of Zhang and Zhang (2002) [30]. Nonlinear methods also include diffusion maps prescribed by Coifman et al. (2005) [4].

PCA and MDS are simple and straight forward algorithms which recover the true underlying structure of the data lying on or near a linear subspace of the high dimensional input space. ISOMAP builds on classical MDS with a goal to preserve all geodesic distances between points. Kernel PCA is an extension of PCA using techniques of kernel methods. Kernel eigenmaps and diffusion maps on the other hand, represents (embeds) the data as a cloud of points in a Euclidean space. They use the eigenvectors of Markov matrices as coordinates of the data expecting to achieve dimensionality reduction by capturing the main structure of the data in a few dimensions [4]. A Markov matrix describes the transitions of a Markov chain, while a Markov chain on a graph is simply a random walk on the vertices of a graph with equal probability for the transition steps taken to any of the neighbours of the current vertex (irrespective of the history of the walk). The Laplacian used in the above algorithm is one such Markov matrix.

The two main advantages of diffusion process based algorithms over classical dimensionality reduction techniques (PCA and MDS) is that they are nonlinear and they preserve local structures. This is important because usually the input data do not lie on a linear manifold. Furthermore, in most cases preserving distances of close points is more meaningful than preserving distances of points far apart. It is important to note also that Coifman and Lafon in [4] interpreted all kernel methods as

special cases of the general framework based on diffusion processes.

Key features of diffusion maps

1. From the algorithm, we see that diffusion maps define an embedding of a high n-dimensional data space onto a low m-dimensional Euclidean space using the set of eigenvectors corresponding to the first few smallest eigenvalues of Markov matrices. However, these embeddings are not unique, but are a multiscale family of geometric embeddings corresponding to descriptions at different scales. Thus there exist a family of diffusion maps. In [19] Nadler et al. argued that different normalisations of the Markov chain on the graph lead to different limiting differential operators. This dissertation focusses on the map that leads to the eigenfunctions of the Laplace Beltrami operator (discussed next) because it is best suited for the analysis of the geometry of data sets regardless of their possible non-uniform density [19].

2. The robustness of this techniques in recovering underlying structures of a manifold is based on the fact that the Laplace-Beltrami operator provides an optimal embedding for the manifold. The Laplace-Beltrami operator is a generalisation of the Laplace operator operating on surfaces and manifolds. In the case of a data set, we assume that the adjacency graph computed from the data points converges to the manifold and the weighted Laplacian of this adjacency graph converges to the Laplace Beltrami operator. The central nature of the Laplace Betrami operator in the heat or diffusion equation is a basis for using the heat kernel for the choice of weights. As a result the embedding of the maps for the data converges to the eigenmaps of the Laplace Beltrami operator defined intrinsically on the entire manifold [1].

3. The implementation of the algorithm is simple and efficiently computable, because it involves one sparse eigenvalue problem (Equation 1.3) and a few local computations, see Appendix 2. However the distance computations in search of neighbouring points is a seemingly difficult task depending on the dimensionality to the data.

4. References [3, 14, 19, 9] showed the close relationship between kernel eigenmaps and diffusion maps and [1] stated the close connection of eigenmaps to spectral clustering. This therefore connects diffusion maps to clustering. This can be

rationalised by the fact that by trying to preserve local information in the embedding diffusion maps and eigenmaps implicitly emphasises the natural cluster in the data set. To demonstrate this relationship, Chapter 5 of this dissertation clearly shows an example how diffusion maps can be used for clustering.

5. Another attractive feature of the algorithm is its robustness to noise and small perturbations of the data. Firstly, due to their locality preserving nature diffusion maps are relatively insensitive to noise. This was numerically demonstrated by Lafon and Lee in [14]. Furthermore, since the algorithm is based on the intrinsic geometric structure of the manifold, it is stable with respect to the embedding. In our example of moving faces in Figure 1.1, different rotations should lead to different input spaces but our algorithm will always recover the same embedding of the underlying manifold independent of the rotations.

6. Finally, diffusion maps have a lot of useful applications. For instance, usually a biological perception organ like the brain is confronted with high dimensional stimuli from which it most recover a low dimensional structure. With diffusion maps the approach to recovering such a low dimensional structure is inherently local, as a result a natural clustering surfaces, leading to the emergence of categories in biological perception. This can be viewed as the implication of the connection of diffusion maps and clustering to human perception and learning which motivated earlier discussions in dimensionality reduction [23, 21, 28, 1].

Chapter 2

THE CHOICE OF WEIGHTS

In this chapter and the following one I present the theoretical justification of the diffusion map algorithm presented in Chapter 1. I will start, in Section 2.1, by showing that the embedding provided by the algorithm is optimally locality preserving in a certain sense. In Section 2.2 I discuss on the choice of the scaling parameter ε.

2.1 Optimal Embedding

The following exposè is based on spectral graph theory. Remember we represented the data set by a weighted graph $G = (V, E)$ with each pair of neighbouring points (nodes) connected by a weighted edge. Let $\mathbf{y} = (y_1, y_2, \ldots, y_k)^T$ be a map of G to a line with connected pair of points staying as close to each other as possible. A "good" map would minimise the objective function

$$\sum_{i,j=1}^{k} (y_i - y_j)^2 W_{ij} \tag{2.1}$$

under appropriate constraints. From the look of things, this objective function maps close points close because it penalises points far apart more. But since W_{ij} is symmetric and $D_{ii} = \sum_{j=1}^{k} W_{ji}$

$$\begin{aligned} \sum_{i,j=1}^{k} (y_i - y_j)^2 W_{ij} &= \sum_{i,j=1}^{k} (y_i^2 - y_j^2 - 2y_i y_j) W_{ij} \\ &= \sum_{i=1}^{k} y_i^2 D_{ii} + \sum_{j=1}^{k} y_j^2 D_{jj} - 2 \sum_{i,j=1}^{k} y_i y_j W_{ij} \\ &= 2\mathbf{y}^T \mathbf{L} \mathbf{y}. \end{aligned}$$

Thus for any $\mathbf{y}$, the following relation holds

$$\frac{1}{2}\sum_{i,j=1}^{k}(y_i - y_j)^2 W_{ij} = \mathbf{y}^T\mathbf{L}\mathbf{y} \tag{2.2}$$

where $\mathbf{L}$, $\mathbf{D}$ and $\mathbf{W}$ are as in Equation (1.3). This also shows that $\mathbf{L}$ is positive semidefinite. Therefore, minimising the objective function now turns into minimising the following:

$$\underset{\{\mathbf{y}|\mathbf{y}^T\mathbf{D}\mathbf{y}=1\}}{\operatorname{argmin}} \quad \mathbf{y}^T\mathbf{L}\mathbf{y} \tag{2.3}$$

where $\mathbf{y}^T\mathbf{D}\mathbf{y} = 1$ is a normalisation that removes the effect of any arbitrary scaling [1].

Lemma 1: *The* $\mathbf{y}$ *that minimises the objective function* (2.1) *is the eigenvector corresponding to the smallest eigenvalue of the generalised eigenvalue problem* (1.3).

Proof of Lemma 1:
Let $(\mathbf{v}_i, \lambda_i), i = 1, 2, \ldots, k$, be the eigenpair of (1.3). Thus $\mathbf{L}\mathbf{v}_i = \lambda_i\mathbf{D}\mathbf{v}_i$. Let us suppose that eigenvectors $\mathbf{v}_i$ are normalised so that $\mathbf{v}_i^T\mathbf{D}\mathbf{v}_i = 1$. Let $\mathbf{y} \in \mathbb{R}^k$ be arbitrary satisfying the normalisation condition $\mathbf{y}^T\mathbf{D}\mathbf{y} = 1$. We can express $\mathbf{y}$ as $\mathbf{y} = \sum_{i=1}^{k} a_i\mathbf{v}_i$ for suitable $a_i \in \mathbb{R}, i = 1, \ldots, k$.

$$\begin{aligned}
\text{Thus} \quad \mathbf{y}^T\mathbf{L}\mathbf{y} &= \left(\sum_{i=1}^{k} a_i\mathbf{v}_i^T\right)\mathbf{L}\left(\sum_{j=1}^{k} a_j\mathbf{v}_j\right) \\
&= \sum_{i=1}^{k} a_i\mathbf{v}_i^T \sum_{j=1}^{k} a_j\mathbf{L}\mathbf{v}_j \\
&= \sum_{i=1}^{k} a_i\mathbf{v}_i^T \sum_{j=1}^{n} a_j\lambda_j\mathbf{D}\mathbf{v}_j \\
&= \sum_{i=1}^{k} \lambda_i a_i^2\mathbf{v}_i^T\mathbf{D}\mathbf{v}_i \\
&= \lambda_1 a_1^2 + \lambda_2 a_2^2 + \ldots + \lambda_k a_k^2.
\end{aligned}$$

Without loss of generality let us assume that the eigenvalues are ordered with λ_1 being the smallest. Then

$$\mathbf{y}^T\mathbf{L}\mathbf{y} \geq \lambda_1(a_1^2 + a_2^2 + \ldots + a_k^2). \tag{2.4}$$

Since we are looking for a minimiser $\mathbf{y}$ which satisfies the normalisation condition $\mathbf{y}^T\mathbf{D}\mathbf{y} = 1$, we have the following restriction on $a_i, i = 1, \ldots, k$:

$$\begin{aligned} 1 &\equiv \mathbf{y}^T\mathbf{D}\mathbf{y} \\ &= \left(\sum_{i=1}^{k} a_i \mathbf{v}_i^T\right) \mathbf{D} \left(\sum_{j=1}^{k} a_j \mathbf{v}_j\right) \\ &= \sum_{i,j=1}^{k} a_i a_j \mathbf{v}_i^T \mathbf{D} \mathbf{v}_j \;=\; \sum_{i=1}^{k} a_i^2. \end{aligned}$$

Consequently (2.4) implies that $\mathbf{y}^T\mathbf{L}\mathbf{y} \geq \lambda_1$. On the other hand, choosing coefficients $a_1 = 1,\ a_2 = a_3 = \ldots = a_k = 0$, such that $\mathbf{y} = \mathbf{v}_1$, we have $\mathbf{y}^T\mathbf{L}\mathbf{y} = \lambda_1$ and $\mathbf{y}^T\mathbf{D}\mathbf{y} = 1$. Thus $\mathbf{y} = \mathbf{v}_1$, is a minimiser of $\mathbf{y}^T\mathbf{L}\mathbf{y}$ subject to the condition $\mathbf{y}^T\mathbf{D}\mathbf{y} = 1$. **End of proof**.

Let $\mathbf{1}$, a vector of ones, be a constant function taking the value one at each vertex. Then $\mathbf{y} = \mathbf{1}$ would be the eigenvector solution with $\lambda = 0$ as can be seen by:

$$\begin{aligned} \mathbf{L1} &= (\mathbf{D} - \mathbf{W})\,\mathbf{1} \\ &= \begin{pmatrix} \sum_{j=1}^{k} W_{1j} & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \sum_{j=1}^{k} W_{kj} \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} - \begin{pmatrix} W_{11} & \ldots & W_{1k} \\ \vdots & \ddots & \vdots \\ W_{k1} & \ldots & W_{kk} \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} \sum_{j=1}^{k} W_{1j} \\ \vdots \\ \sum_{j=1}^{k} W_{kj} \end{pmatrix} - \begin{pmatrix} \sum_{j=1}^{k} W_{1j} \\ \vdots \\ \sum_{j=1}^{k} W_{kj} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} = 0\mathbf{D1}. \end{aligned}$$

This trivial solution would map all the vertices of G to the point 1 on the real line. To avoid this we add an orthogonality constraint, i.e. the appropriate minimisation problem to solve is

$$\underset{\substack{\mathbf{y}^T\mathbf{L}\mathbf{y}=1 \\ \mathbf{y}^T\mathbf{D}\mathbf{1}=0}}{\operatorname{argmin}}\ \mathbf{y}^T\mathbf{L}\mathbf{y}. \tag{2.5}$$

Its solution is the eigenvector corresponding to the smallest non-zero λ.

A similar argument goes for a general problem of embedding. Say we want to embed a graph into an m-dimensional Eulidean space. This time the embedding will be a $k \times m$ matrix $\mathbf{Y} = \begin{bmatrix}\mathbf{y}_1 & \mathbf{y}_2 & \dots & \mathbf{y}_m\end{bmatrix}$ where the embedding coordinates of the i^{th} vertex corresponds to the i^{th} row of $\mathbf{Y}$. The objective function to minimise now becomes

$$\sum_{i,j=1}^{k} \|\mathbf{y}^{(i)} - \mathbf{y}^{(j)}\|^2 W_{ij} = 2\text{tr}(\mathbf{Y}^T\mathbf{L}\mathbf{Y}). \tag{2.6}$$

where $\mathbf{y}^{(i)} = [y_1^{(i)}, y_2^{(i)}, \dots, y_m^{(i)}]^T$ is the m-dimensional representation of the i^{th} vertex. The Formula 2.6 can be verified as follows:

$$\begin{aligned}
\sum_{i,j=1}^{k} \|\mathbf{y}^{(i)} - \mathbf{y}^{(j)}\|^2 W_{ij} &= \sum_{i,j=1}^{k} \|(y_1^{(i)}, y_2^{(i)}, \dots, y_m^{(i)})^T - (y_1^{(j)}, y_2^{(j)}, \dots, y_m^{(j)})^T\|^2 W_{ij} \\
&= \sum_{i,j=1}^{k} \big[(y_1^{(i)} - y_1^{(j)})^2 + (y_2^{(i)} - y_2^{(j)})^2 + \dots + (y_m^{(i)} - y_m^{(j)})^2\big] W_{ij} \\
&= \sum_{i,j=1}^{k} \big[(y_1^{(i)})^2 - 2y_1^{(i)} y_1(j) + \dots - 2y_m^{(i)} y_m^{(j)} + (y_m^{(j)})^2\big] W_{ij} \\
&= \sum_{i=1}^{k} y_{1i}^2 D_{ii} + \sum_{j=1}^{k} y_{1j}^2 D_{jj} - 2\sum_{ij} y_{1i} y_{1j} W_{ij} + \dots + \\
&\quad \sum_{i=1}^{k} y_{ki}^2 D_{ii} + \sum_{j=1}^{k} y_{kj}^2 D_{jj} - 2\sum_{ij=1}^{k} y_{ki} y_{kj} W_{ij} \\
&= 2\mathbf{y}_1\mathbf{L}\mathbf{y}_1 + 2\mathbf{y}_2\mathbf{L}\mathbf{y}_2 + \dots + 2\mathbf{y}_k\mathbf{L}\mathbf{y}_k \\
&= 2\text{tr}(\mathbf{Y}^T\mathbf{L}\mathbf{Y}).
\end{aligned}$$

Note the switch to subscripts in the 4^{th} and 5^{th} lines. This is just to related it to the earlier notation used for $\mathbf{D}$ and $\mathbf{W}$. Now, minimising the objective function (2.6) reduces to finding

$$\underset{\mathbf{Y}^T\mathbf{D}\mathbf{Y}=\mathbf{I}}{\text{argmin}}\ \text{tr}(\mathbf{Y}^T\mathbf{L}\mathbf{Y}). \tag{2.7}$$

In this case, the constraints $\mathbf{Y}^T\mathbf{D}\mathbf{Y} = \mathbf{I}$ prevents a collapse onto a subspace of dimension less than m [1].

Lemma 2: *The solution of* (2.7) *is given by the columns of* $\mathbf{Y}$ *which are eigenvectors corresponding to the lowest eigenvalues of the generalised eigenvalue problem* (1.3).

Proof of Lemma 2:
Let $(\mathbf{v}_i, \lambda_i), i = 1, 2, \dots, k$, be the eigenpair of (1.3). Thus $\mathbf{L}\mathbf{v}_i = \lambda_i\mathbf{D}\mathbf{v}_i$. Let us suppose that eigenvectors $\mathbf{v}_i$ are normalised so that $\mathbf{v}_i^T\mathbf{D}\mathbf{v}_i = 1$. Let $\mathbf{Y} \in \mathbb{R}^{k\times m}$ be a

matrix satisfying the normalisation condition $\mathbf{Y}^T\mathbf{D}\mathbf{Y} = \mathbf{I}$. Denoting columns of $\mathbf{Y}$ as $\mathbf{y}_j$, $j = 1, \ldots, m$, we have

$$\mathbf{y}_j = \sum_{i=1}^{k} a_{ji}\mathbf{v}_i \qquad \text{for suitable} \quad a_{ji} \in \mathbb{R},\ j = 1, \ldots, m,\ i = 1, \ldots, k.$$

Thus

$$\mathbf{Y} = \left[\sum_{i=1}^{k} a_{1i}\mathbf{v}_i \ \sum_{i=1}^{k} a_{2i}\mathbf{v}_i \ \cdots \ \sum_{i=1}^{k} a_{mi}\mathbf{v}_i \right].$$

We have

$$\begin{aligned}
\operatorname{tr}(\mathbf{Y}^T\mathbf{L}\mathbf{Y}) &= \operatorname{tr}\left(\left[\sum_{j=1}^{k} a_{1j}\mathbf{v}_j \ \cdots \ \sum_{j=1}^{k} a_{mj}\mathbf{v}_j \right]^T \mathbf{L} \left[\sum_{i=1}^{k} a_{1i}\mathbf{v}_i \ \cdots \ \sum_{i=1}^{k} a_{mi}\mathbf{v}_i \right] \right) \\
&= \operatorname{tr}\left(\left[\sum_{j=1}^{k} a_{1j}\mathbf{v}_j \ \cdots \ \sum_{j=1}^{k} a_{mj}\mathbf{v}_j \right]^T \left[\sum_{i=1}^{k} a_{1i}\mathbf{L}\mathbf{v}_i \ \cdots \ \sum_{i=1}^{k} a_{mi}\mathbf{L}\mathbf{v}_i \right] \right) \\
&= \sum_{i,j=1}^{k} a_{1j}a_{1i}\mathbf{v}_j\mathbf{L}\mathbf{v}_i + \cdots + \sum_{i,j=1}^{k} a_{mj}a_{mi}\mathbf{v}_j\mathbf{L}\mathbf{v}_i \\
&= \sum_{i,j=1}^{k} a_{1j}a_{1i}\lambda_i\mathbf{v}_j\mathbf{D}\mathbf{v}_i + \cdots + \sum_{i,j=1}^{k} a_{mj}a_{mi}\lambda_i\mathbf{v}_j\mathbf{D}\mathbf{v}_i.
\end{aligned}$$

Which implies

$$\operatorname{tr}(\mathbf{Y}^T\mathbf{L}\mathbf{Y}) = \sum_{i=1}^{k} \lambda_i a_{1i}^2 + \ldots + \sum_{i=1}^{k} \lambda_i a_{mi}^2. \tag{2.8}$$

We are looking for a minimiser $\mathbf{Y}$ satisfying the normalisation condition $\mathbf{Y}^T\mathbf{D}\mathbf{Y} = \mathbf{I}$, which implies

$$\begin{aligned}
\mathbf{I} &\equiv \mathbf{Y}^T\mathbf{D}\mathbf{Y} \\
&= \left[\sum_{j=1}^{k} a_{1j}\mathbf{v}_j \ \cdots \ \sum_{j=1}^{k} a_{mj}\mathbf{v}_j \right]^T \left[\sum_{i=1}^{k} a_{1i}\mathbf{D}\mathbf{v}_i \ \cdots \ \sum_{i=1}^{k} a_{mi}\mathbf{D}\mathbf{v}_i \right] \\
&= \begin{pmatrix}
\sum_{i=1}^{k} a_{1i}^2 & \sum_{i=1}^{k} a_{1i}a_{2i} & \cdots & \sum_{i=1}^{k} a_{1i}a_{mi} \\
\sum_{i=1}^{k} a_{2i}a_{1i} & \sum_{i=1}^{k} a_{2i}^2 & \cdots & \sum_{i=1}^{k} a_{2i}a_{mi} \\
\vdots & \vdots & \cdots & \vdots \\
\sum_{i=1}^{k} a_{mi}a_{1i} & \sum_{i=1}^{k} a_{mi}a_{2i} & \cdots & \sum_{i=1}^{k} a_{mi}^2
\end{pmatrix}.
\end{aligned}$$

Therefore each of the diagonal elements is equal to 1 and each of the off-diagonal elements is equal to 0. Consequently using the cyclic property of the trace of square matrices and trace inequality for matrix products we have

$$\mathrm{tr}(\mathbf{Y}^T\mathbf{L}\mathbf{Y}) \geq \lambda_1 + \lambda_2 + \ldots + \lambda_m.$$

However, choosing $a_{jj} = 1$, and $a_{ji} = 0$ for $i \neq j$ such that $\mathbf{Y} = \left[\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m\right]$ will give

$$\mathrm{tr}(\mathbf{Y}^T\mathbf{L}\mathbf{Y}) = \lambda_1 + \lambda_2 + \cdots + \lambda_m.$$

Thus $\mathbf{Y} = \left[\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m\right]$ is a minimiser of $\mathrm{tr}(\mathbf{Y}^T\mathbf{L}\mathbf{Y})$ subject to the condition $\mathbf{Y}^T\mathbf{D}\mathbf{Y} = \mathbf{I}$.
End of proof.

2.2 The Choice of ε

In the diffusion map algorithm, the choice of the parameter ε is very crucial in the computation of the weights $W_{ij} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\varepsilon}\right)$. Physically it could be interpreted as a scale parameter in the heat kernel [13, 3]. In fact to be precise the scale is $\sqrt{\varepsilon}$. It must be said that the choice of ε is data dependent. In the numerical implementation of the algorithm, I have experience the effect of an inappropriate choice of ε. I have got numerous instances when my algorithm could not converge due to the value of ε I used. This has further motivated me to discuss this issue.

Different schemes have been proposed with regards to an appropriate choice of ε. It must be said the value we are looking for is not a single one but a range of values. So even though the different schemes might give different values but the values usually fall within the accepted range for a given data set. Lafon in [13] choose ε to be the order of the average smallest non-zero value of $\|\mathbf{x}_i - \mathbf{x}_j\|^2$, that is

$$\varepsilon = \frac{1}{k}\sum_{i=1}^{k} \min_{j:\mathbf{x}_j \neq \mathbf{x}_i} \|\mathbf{x}_i - \mathbf{x}_j\|^2. \tag{2.9}$$

It would seem that the main motivation for this choice of value for ε is that it ensures that some diffusion will take place and that you are sure that the diffusion is far from conclusion.

However, the scheme I have been using is the one proposed by Singer et al. in [25]. It has an easier physical interpretation and its implementation numerically gives a more convincing result. The justification of their argument is that when ε is relatively too large compared to $\|\mathbf{x}_i - \mathbf{x}_j\|^2$ then the entries of the weight matrix $\mathbf{W}$ will be very

close to one while a relatively small ε will give almost zero entries for $\mathbf{W}$. The former means most of the diffusion has taken place while the latter implies very little diffusion took place. Both scenarios are not very interesting for the purpose of diffusion map applications. Values of ε that would arouse interest therefore lie between these two extremes. Based on this idea they proposed the following scheme:

1. Construct a sizeable ε-dependent weight matrix $\mathbf{W} \equiv \mathbf{W}(\varepsilon)$ by (1.1) for several values of ε.

2. Compute
$$L(\varepsilon) = \sum_{i=1}^{k} \sum_{j=1}^{k} W_{ij}(\varepsilon) \tag{2.10}$$

3. Plot $L(\varepsilon)$ using a logarithmic plot. This plot will have two asymptotes when $\varepsilon \to 0$ and $\varepsilon \to \infty$.

4. Choose ε where the logarithmic plot of $L(\varepsilon)$ appears linear.

I would remark that ε-dependent weight matrix $\mathbf{W} \equiv \mathbf{W}(\varepsilon)$ should not be arbitrary but a sample of the data set being worked on. They implemented the above scheme in [25].

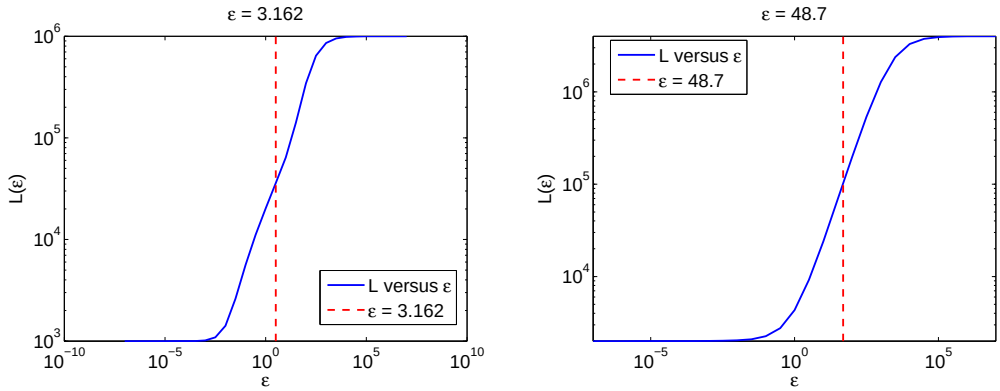


Figure 2.1: Logarithmic plots of $L(\varepsilon)$ against ε, 1000×1000 weight matrix from the data set of the second illustrative example in Section 1.2.2 (left), 2000×2000 weight matrix from the data set of the 2D manifold in Section 4.3 (right).

For each of the data sets I used in this dissertation I implemented this scheme to get an appropriate ε before I implemented the diffusion map algorithm. Figure 2.1 and 2.2 show a sample out the ε values I used in this project. The intersection of the red dotted line and the blue curve of L versus ε gives the value of ε I used. The red line is drawn after first observing the shape of the blue curve and identifying an

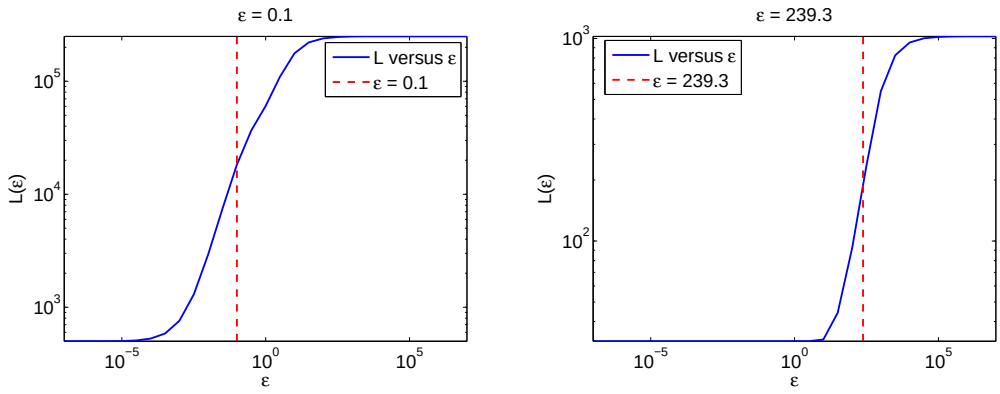


Figure 2.2: Logarithmic plots of $L(\varepsilon)$ against ε, 500×500 weight matrix from the data set of the "circular" clusters of Section 5.1.1 (left), 32×32 weight matrix from the data set of the set of faces of Section 5.2 (right).

appropriate value of ε corresponding to some point along the approximately straight line between the two asymptotes. The different ε values in these plots confirmed the earlier assertion I made concerning the dependence of this value on the data set.

Chapter 3

RANDOM WALK AND THE HEAT EQUATION

In this chapter, I use the relationship between random walk on graphs and the heat (diffusion) equation to justify the use of the first few eigenvectors of $\mathbf{L}$ for the low dimensional representation. The data is modelled by a weighted graph. I will show that the random walk on the graph is in some sense equivalent to solving the heat equation in a suitable domain. Furthermore, the solution of the heat equation can be approximated using the first few eigenvectors of the heat equation which correspond to the first few largest eigenvalues of the system derived from trying to solve the heat equation on the given domain. Consequently, instead of doing a random walk on the domain or solving the heat equation on the domain which can be difficult tasks sometimes, especially if the domain is complex and large, we directly compute the eigenvectors and eigenvalues which is equivalent to the diffusion map algorithm.

Assume that we want to find out the structure of a landscape. One crude way of doing this would be gathering a group of soldiers or students for instance and allow then to wonder about all from the same starting point. After some time it is highly likely that the locations most visited are closer to the starting point than those less visited. Similarly, given a complex high dimensional data set one can start from a point and diffuse (do a random walk) in the data to discover its underlying structure through the connectivity of the points. This leads to a very important concept which is *diffusion distance*. The diffusion distance between two points is considered to be small if they are connected by paths containing edges with high weights, and otherwise large. So diffusion distance reflects the connectivity of the points within a data set and the connectivity of the data points in a diffusion process in turn reflects the intrinsic geometry of the data set [14, 3]. It could be noted that in the absence of geometry, diffusion distances correspond to Euclidean distances.

Now we demonstrate both theoretically and numerically how the random walk on graph averaged over time is equivalent to the heat equation involving the probability density function associated with the positions on a random walk. I start with a special graph represented in 2D which is a square lattice. Then I study random walk on a slightly more complicated ("n-shape") domain.

3.1 Theoretical derivation of the heat equation

Let us consider a rectangular lattice as in Figure 3.1. We identify the nodes of the grid as the vertices of the graph while the lines connecting the nodes can be viewed as edges with the weights shown. The mesh point $(x_\alpha, y_\beta) = (\alpha\Delta x, \beta\Delta x)$ for $\alpha = 0, 1, \ldots, N_x$ and $\beta = 0, 1, \ldots, N_y$ is associated with the i^{th} vertex of the graph where $i = \alpha(N_y+1)+\beta+1$ and the order of the weight matrix $\mathbf{W}$, $k = (N_x+1)\times(N_y+1)$.

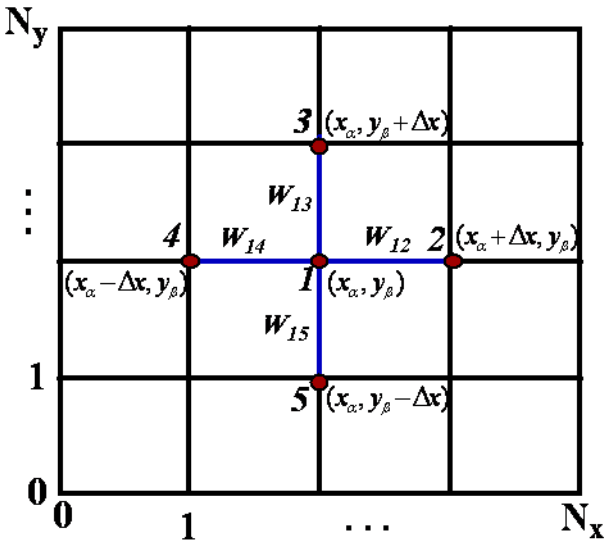


Figure 3.1: A rectangular lattice with weights along edges showing the *5-point stencil* [5] of equation 3.3.

We consider the following random walk on a lattice (x_α, y_β) for $\alpha = 0, 1, \ldots, N_x$ and $\beta = 0, 1, \ldots, N_y$. At each time step Δt a random walker can stay at its current position (x_α, y_β) or jump from its current position to one of the neighbouring mesh points $(x_\alpha, y_\beta + \Delta x)$, $(x_\alpha + \Delta x, y_\beta)$, $(x_\alpha, y_\beta - \Delta x)$, or $(x_\alpha - \Delta x, y_\beta)$. The transition probability from node i at time t to an adjacent node j at time $t+\Delta t$ is proportional to the weights W_{ij}. In what follows, we assume that $W_{ij} = 4d$, i.e. W_{ij} are equal to the same constant if j is a neighbour of i, otherwise it is 0. The transition probability

from node i at time t to an adjacent node j at time $t+\Delta t$ is $\frac{d\Delta t}{\Delta x^2}$ while the probability of staying at the same node is given by $(1-4\frac{d\Delta t}{\Delta x^2})$ [18, 19].

The probability of being at node i at time $(t+\Delta t)$ is the sum of the probabilities of being at points (x_α, y_β), $(x_\alpha, y_\beta+\Delta x)$, $(x_\alpha+\Delta x, y_\beta)$, $(x_\alpha, y_\beta-\Delta x)$, and $(x_\alpha-\Delta x, y_\beta)$ at time t. This means:

$$\begin{aligned} p(x_\alpha, y_\beta; t+\Delta t) &= (1-4\frac{d\Delta t}{\Delta x^2})p(x_\alpha, y_\beta; t) + \frac{d\Delta t}{\Delta x^2}p(x_\alpha, y_\beta+\Delta t; t) \\ &\quad +\tfrac{d\Delta t}{\Delta x^2}p(x_\alpha, y_\beta-\Delta t; t) + \frac{d\Delta t}{\Delta x^2}p(x_\alpha+\Delta x, y_\beta; t) \\ &\quad +\frac{d\Delta t}{\Delta x^2}p(x_\alpha-\Delta x, y_\beta; t). \end{aligned} \tag{3.1}$$

Let us define the discretised Laplace operator as

$$\Delta_{\Delta x}p_(x_\alpha, y_\beta) = \frac{p(x_\alpha, y_\beta+\Delta x)+p(x_\alpha+\Delta x, y_\beta)+p(x_\alpha, y_\beta-\Delta x)+p(x_\alpha-\Delta x, y_\beta)-4p(x_\alpha, y_\beta)}{\Delta x^2}$$

with the t dropped from the argument of p [5]. Thus Equation 3.1 can be rewritten as:

$$p(x_\alpha, y_\beta; t+\Delta t) - p(x_\alpha, y_\beta; t) = \Delta t d(\Delta_{\Delta x}p)(x_\alpha, y_\beta). \tag{3.2}$$

Dividing by Δt and taking limits as $\Delta x \to 0$ then as $\Delta t \to 0$ we have

$$\frac{\partial p}{\partial t} = d\Delta p \tag{3.3}$$

which is the heat equation in terms of probabilities and d is the diffusion coefficient. Since the random walk does not go beyond the boundaries we have Neumann boundary conditions for Equation 3.3 given by

$$\frac{\partial p}{\partial \mathbf{n}} = 0 \tag{3.4}$$

where $\mathbf{n}$ is the outwards normal to the boundary. Thus the random walk on the graph can be approximately described by the heat equation with Neumann boundary conditions.

3.2 Numerical findings

3.2.1 Square grid

Now setting an arbitrary grid of size $N \times N$ with $N = 50$, I started my random walk at the point $(x_\alpha, y_\beta) = (N/2, N/2)$ in the square grid $[0,1] \times [0,1]$. Therefore

$\Delta x = 1/N = 0.02$. I used $d = 0.2499$. Guided by the fact that I want the transition probabilities to be small I then chose $\Delta t = 0.04\Delta x^2 = 1.6 \times 10^{-5}$. Thus giving me a transition probability $p = \frac{d\Delta t}{\Delta x^2} = 1\%$. I ran the random walk forward for a total time of $T = 0.01$ for 10^6 realisations. Taking the mean of these random walks by dividing by the total number of realisations, I plotted the results in the first panel of Figure 3.2. Note that I plotted only the middle portion of the 50 by 50 grid to show a bigger picture, since most of the random walk was around the midpoint of the grid.

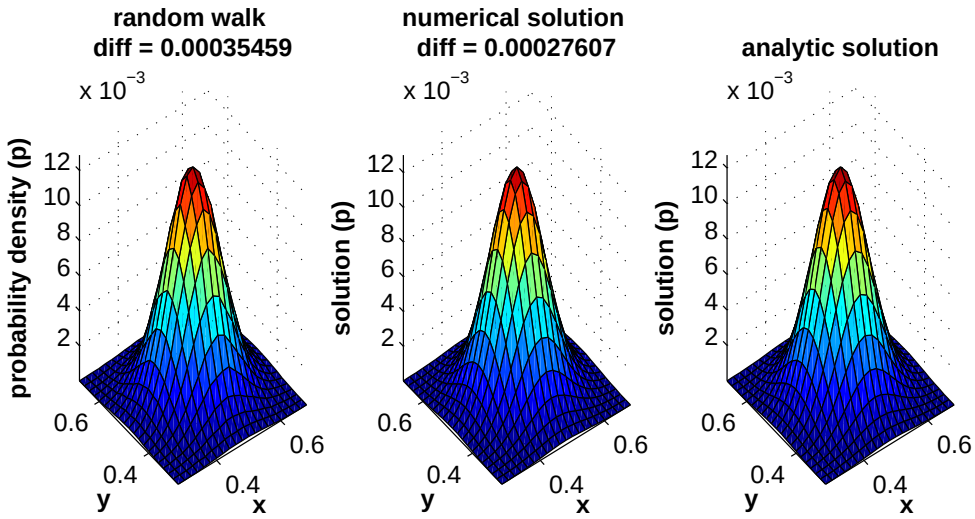


Figure 3.2: A surface plot of the random walk, analytic and numerical solution of the heat equation.

Furthermore, an analytic solution of Equation (3.3) in an infinite domain for random walk starting at $\mathbf{r}_0 = (x_0, y_0)$ at time $t = 0$, is the Gaussian probability density function associated with the position vector of the random walk $\mathbf{r} = (x, y)$ and is given by:

$$p(\mathbf{r}, t) = \frac{1}{4\pi dt} \exp\left[-\frac{(x - x_0)^2 + (y - y_0)^2}{4Dt}\right] \tag{3.5}$$

where d is the diffusivity and t represents time. This is plotted with $t = T = 0.01$, in the middle panel. It is important to note that Equation 3.5 does not satisfy the boundary conditions (3.4). But because at an intermediate time the diffusion is far from the boundary, its plot will do for comparison purposes.

Finally, I used finite differences to solve the heat equation (3.3) on the domain $[0, 1] \times [0, 1]$ with the boundary condition (3.4) and initial condition $p(N/2, N/2) = 1$. A plot of the result is in the right panel. The difference (`diff`) displayed in the first two panels is the maximum difference (the infinity norm of the discrepancy) of the

respective solutions from the analytic solution. This shows how close the solutions are. 2D plots of these same results in the same order respectively are given in Figure 3.3.

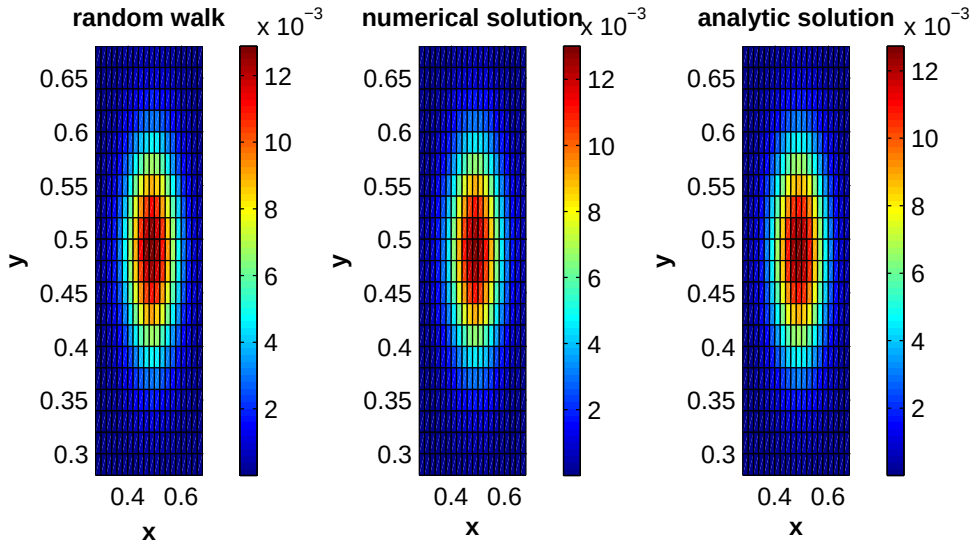


Figure 3.3: A 2D plot of the random walk, analytic and numerical solution of the heat equation.

We see a clear agreement in these numerical results (the differences show how close these results are). This, therefore, confirms our earlier claim about the relationship between random walk and the heat equation. In other words, if a random walk is started at node (x_α, y_β) at time $t = 0$ then the probability density function associated with the position vector of this walk at some time $t > 0$ will be a solution of the heat equation.

3.2.2 "n-shape" domain

Similar to the above presentation on a square grid I further investigate the diffusion on an "n-shape" (see shape in Figure 3.4) domain through the random walk and the solution of the heat equation on the domain. At this juncture, I would like to comment on the diffusion distance. In the square grid discussed previously there were no geometric constrains since the diffusion never got to the boundary. As a result the diffusion distance is equivalent to the Euclidean distance. However, in the n-shape domain this is not the case. For example the Euclidean distance from point A to either point B or C is the same on the diagram in the left panel of Figure 3.4. But the diffusion distances are not the same. Since it is easier to diffuse from A to B than from A to C, the diffusion distance AB is shorter than the diffusion distance AC.

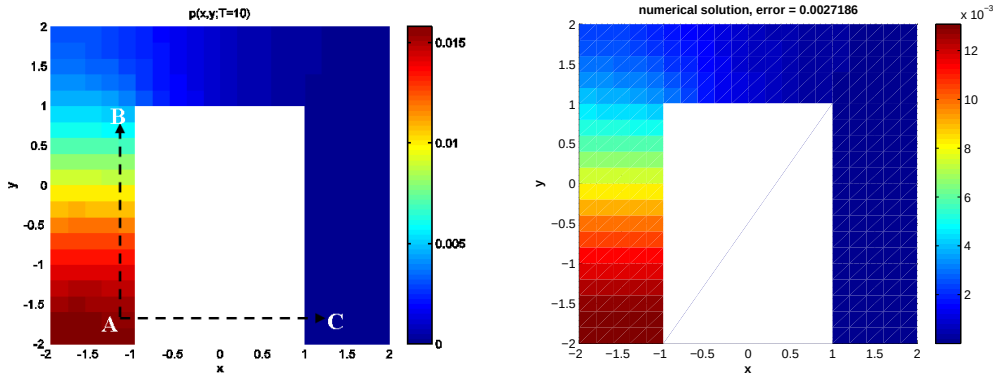


Figure 3.4: (left) random walk on the n-shape domain and (right) numerical solution of the heat equation on this domain.

This time I start the random walk at the point with coordinates $(-2, -2)$. I used $d = 0.2047$, $\Delta t = 0.04\Delta x^2 = 1.6 \times 10^{-5}$, and the transition probability $p = \frac{d\Delta t}{\Delta x^2} = 0.0082 \approx 1\%$. I ran the random walk forward for period of $T = 10$ for 10^7 realisations. Taking the mean out of the total realisations the result is plotted in the left panel of Figure 3.4.

On the other hand, using finite differences to solve the heat equation (3.3) with its boundary condition (3.4) over the n-shape domain with an initial value of a delta function taking a value of 1 at the coordinate $(-2, -2)$ I plotted that result in the right panel of Figure 3.4. Both plots are surface contour plots. The profile of the diffusion is depicted by these contours with colour variations from red (the highest value) to dark blue (the lowest), see colour bar in figure. The "error" (which is a misnomer) on the right panel of Figure 3.4 is the maximum difference (L^∞ norm) between the two solutions. It further confirms how close these two profiles are. Therefore, again one can confidently say that doing a random walk on a domain and averaging over the total number of realisations is the same as solving the heat equation involving the transition probabilities of the Markov chain on the graph composing the domain.

3.2.3 Solution of a linear system

The probability of being at vertex i at time $t+\Delta t$ is the probability of being at vertex (x_α, y_β) or $(x_\alpha, y_\beta + \Delta x)$ or $(x_\alpha + \Delta x, y_\beta)$ or $(x_\alpha, y_\beta - \Delta x)$ or $(x_\alpha - \Delta x, y_\beta)$ at time t. This led to Equation 3.1 which can be simplified as

$$\begin{aligned} p(x_\alpha, y_\beta; t+\Delta t) &= p(x_\alpha, y_\beta; t) + \frac{\Delta t}{\Delta x^2}\left(dp(x_\alpha, y_\beta + \Delta x; t) + dp(x_\alpha, y_\beta - \Delta x; t)\right. \\ &\quad \left. + dp(x_\alpha + \Delta x, y_\beta; t) + dp(x_\alpha - \Delta x, y_\beta; t) - 4dp(x_\alpha, y_\beta; t)\right). \end{aligned} \tag{3.6}$$

Indexing according to the vertex and taking into account the fact that

$$W_{i,j} = \begin{cases} 4d & \text{when } j \text{ is a neighbour of } i; \\ 0 & \text{otherwise.} \end{cases}$$

Equation (3.6) can be rewritten as

$$p_i(t+\Delta t) = p_i(t) + \frac{\Delta t}{4\Delta x^2}\left(\sum_{j=1}^{k} W_{ij}p_j(t) - \sum_{j=1}^{k} W_{ij}p_i(t)\right) \tag{3.7}$$

where $p_i(t) = p(x_\alpha, y_\beta; t)$ for $i = \alpha(N_y+1)+\beta+1$. Let us denote $\mathbf{p}(t) = (p_1(t), p_2(t), \ldots, p_k(t))^T \in \mathbb{R}^k$. Then (3.7) becomes

$$\begin{aligned} \mathbf{p}(t+\Delta t) &= \mathbf{I}\mathbf{p}(t) + \frac{\Delta t}{4\Delta x^2}\left(\mathbf{W}\mathbf{p}(t) - \mathbf{D}\mathbf{p}(t)\right) \\ &= \left(\mathbf{I} + \frac{\Delta t}{4\Delta x^2}\left(\mathbf{W}-\mathbf{D}\right)\right)\mathbf{p}(t) \\ &= \left(\mathbf{I} - \frac{\Delta t}{4\Delta x^2}\mathbf{L}\right)\mathbf{p}(t). \end{aligned} \tag{3.8}$$

Let $\mathbf{Q} = \mathbf{I} - \dfrac{\Delta t}{4\Delta x^2}\mathbf{L}$. Then Equation 3.8 becomes, for $t = 0$:

$$\mathbf{p}(\Delta t) = \mathbf{Q}\mathbf{p}(0).$$

Using (3.8) again we have

$$\mathbf{p}(2\Delta t) = \mathbf{Q}\mathbf{p}(\Delta t) = \mathbf{Q}^2\mathbf{p}(0).$$

Letting $r = \frac{T}{\Delta t}$ where T is the final time and using mathematical induction we get

$$\mathbf{p}(T) = \mathbf{Q}^r\mathbf{p}(0). \tag{3.9}$$

Let $\overline{\lambda}_i, \mathbf{v}_i$, $i = 1, \ldots, k$ be the eigenvalues and normalised eigenvectors of $\mathbf{Q}$, i.e. $\mathbf{Q}\mathbf{v}_i = \overline{\lambda}_i\mathbf{v}_i$. We can express $\mathbf{p}(0)$ as $\mathbf{p}(0) = \sum_{i=1}^{k} a_i\mathbf{v}_i$ where the coefficients a_j, $j = 1, \ldots, k$ of this basis functions can be determined by

$$a_j = \langle \mathbf{v}_j, \mathbf{p}(0)\rangle = \left\langle \sum_{i=1}^{k} a_i\mathbf{v}_i, \mathbf{v}_j \right\rangle = \sum_{i=1}^{k} a_i \langle \mathbf{v}_i, \mathbf{v}_j\rangle .$$

So from Equation 3.9

$$\begin{aligned} \mathbf{p}(T) &= \mathbf{Q}^r \sum_{i=1}^{k} a_i\mathbf{v}_i = \sum_{i=1}^{k} a_i\mathbf{Q}^r\mathbf{v}_i = \sum_{i=1}^{k} a_i\overline{\lambda}_i^r\mathbf{v}_i \\ &= a_1\overline{\lambda}_1^r\mathbf{v}_1 + a_2\overline{\lambda}_2^r\mathbf{v}_2 + a_3\overline{\lambda}_3^r\mathbf{v}_3 + a_4\overline{\lambda}_4^r\mathbf{v}_4 + \ldots \end{aligned} \tag{3.10}$$

The eigenvalues and eigenvectors are in descending order. The first term corresponds to the trivial eigenvalue $\overline{\lambda} = 1$ and thus the rest of the eigenvalues are less than 1 and raised to the power $\frac{T}{\Delta t}$ they are decaying very rapidly. For these reasons the series can be truncated after the first few terms. Apart from the extreme case when the time T is almost zero, the first few terms will make a good approximation for $\mathbf{p}(T)$. When the time is too large all the terms of right hand side of Equation 3.10 would be numerically zero, the only significant term will be the first term. If on the other hand, the time is small, almost all the terms will be significant and it would be difficult to truncate the series. But if we are looking at an intermediate time then the first few term of the right hand side of Equation 3.10 will be enough to approximate $\mathbf{p}(T)$ as afore-said.

I used $\Delta x = 0.2$, $\Delta t = 0.04\Delta x^2$, $T = 10$ and $d = 0.0082$, which corresponds to $\varepsilon = 0.2$ and are the same values as in the random walk and heat solution of the n-shape domain above. This gives $\overline{\lambda}_6^r \approx 0.0083$, where $\overline{\lambda}_6$ is the sixth eigenvalue. So truncating the series at $\overline{\lambda}_6$ leaves out less than 1% of the value of $\mathbf{p}(T)$. Solving the system (3.9), I plotted the results in Figure 3.5 using the first six eigenvalues and eigenvectors. Which is a good approximation of both the random walk and the finite difference solutions in Figure 3.4. This is confirmed by the "error" indicated in the plot which is again the maximum difference between the solution of the system and the random walk. The maximum discrepancy (L^∞ norm of the difference) between the finite difference solution and the solution of the system is 2.5027×10^{-5} which compares well.

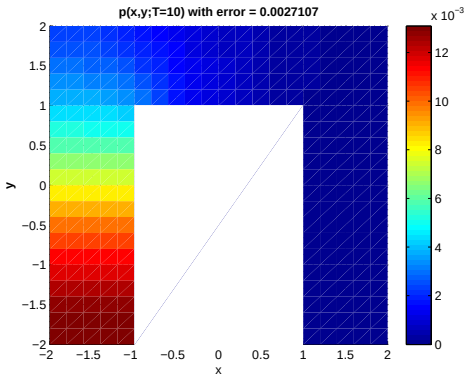


Figure 3.5: A plot of $\mathbf{p}$ at time $T = 10$.

These first few eigenvectors are the diffusion map. This can be shown more rigorously as follows. Remember

$$\mathbf{Q} = \mathbf{I} - c\mathbf{L},$$

where $c = \dfrac{\Delta t}{4\Delta x^2}$. Therefore,

$$\begin{aligned} & \mathbf{Q}\mathbf{v} &=& \overline{\lambda}\mathbf{v} \\ \Leftrightarrow\ & (\mathbf{I} - c\mathbf{L})\,\mathbf{v} &=& \overline{\lambda}\mathbf{v} \\ \Leftrightarrow\ & c\mathbf{L}\mathbf{v} &=& \mathbf{v} - \overline{\lambda}\mathbf{v} \\ \Leftrightarrow\ & \mathbf{L}\mathbf{v} &=& \frac{(1-\overline{\lambda})}{c}\mathbf{v}. \end{aligned}$$

This further implies $\lambda = \frac{(1-\overline{\lambda})}{c}$ are the eigenvalues of $\mathbf{L}$ and since they are in ascending order (refer to the diffusion map algorithm in Section 1.1.1) from $\lambda = 0$ then the eigenvalues of $\mathbf{Q}$ will be ordered in descending order from $\overline{\lambda} = 1$ as shown above. But, the eigenvectors are exactly the same.

So therefore the diffusion map algorithm is a valid approximation of the solution of the probabilistic heat equation on a given domain, while the probabilistic heat equation in turn is equivalent, in the sense discussed above, to the random walk on that same domain. Moreover, the random walk on the domain recovers the connectivity of points on the domain and this eventually describes the underlying structure of the domain hence achieving dimensionality reduction.

3.3 Order of eigenvectors

I have given a probabilistic interpretation of the diffusion algorithm leading to a PDE. Now I want to investigate the behaviour of the eigenvectors in a 2D rectangular manifold embedded in a 3D data space. As shown in the Introduction, Section 1.2, the algorithm will recover the most significant dimension by the second (first non-trivial) eigenvector but I would show which eigenvector represents the second significant dimension. The motivation for this investigation stemmed from a discussion with I. Kevrekidis [11].

To do this I use the eigenvalues and eigenvectors of the Laplace operator in the given domain. The left panel of Figure 3.6 is an example of a data set in 3D which in reality is in 2D. In essence the data could have been represented in 2D using ξ and η coordinate system as in Figure 3.7. First we look for the eigenvalues and eigenvectors of the Laplace operator. That is

$$\begin{aligned} \Delta V_{p,q}(\xi,\eta) &= \lambda_{p,q} V_{p,q} \quad \text{in } (0, L_\xi) \times (0, L_\eta) \\ \text{with } \frac{\partial V_{p,q}}{\partial \mathbf{n}} &= 0 \quad \text{on the boundary of } (0, L_\xi) \times (0, L_\eta). \end{aligned} \tag{3.11}$$

where $p, q \in \mathbb{Z}_0^+$. The $V_{p,q}$ that satisfies the boundary value problem (3.11) are

$$V_{p,q} = \cos\left(\frac{p\pi}{L_\xi}\xi\right)\cos\left(\frac{q\pi}{L_\eta}\eta\right).$$

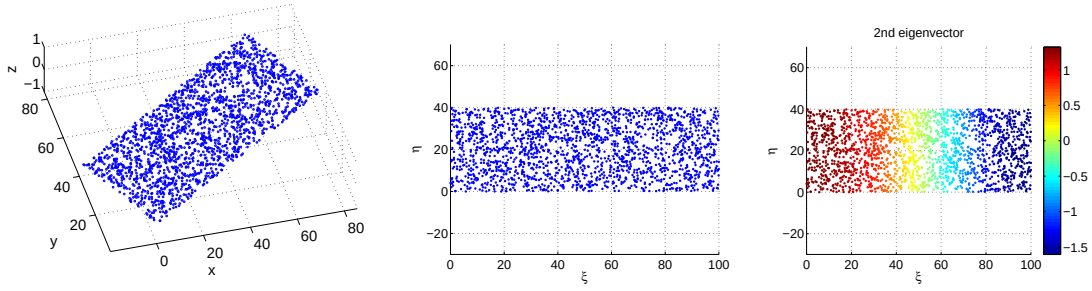


Figure 3.6: A 2D manifold in a 3D space, (left) original data, (middle) a representation in 2D ξ and η coordinate system and (right) coloured according to 2^{nd} eigenvector of the diffusion map.

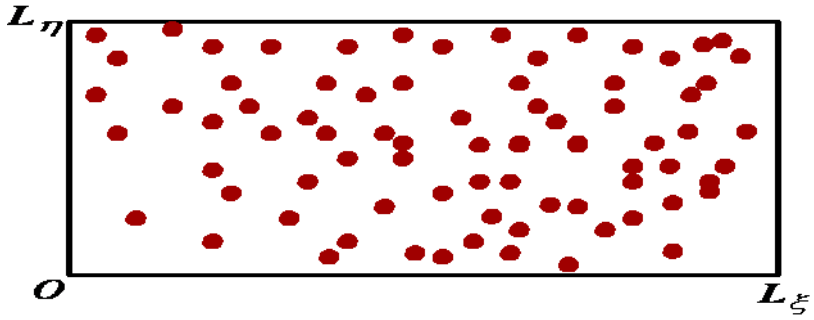


Figure 3.7: A schematic of the 2D manifold in a ξ and η coordinate system.

Therefore

$$\begin{aligned}\Delta V_{p,q}(\xi,\eta) &= \left(-\frac{p^2\pi^2}{L_\xi^2}-\frac{q^2\pi^2}{L_\eta^2}\right)\cos\left(\tfrac{p\pi}{L_\xi}\xi\right)\cos\left(\tfrac{p\pi}{L_\eta}\eta\right)\\ &= -\pi^2\left(\frac{p^2}{L_\xi^2}+\frac{q^2}{L_\eta^2}\right)V_{p,q}\end{aligned}$$

which implies

$$\lambda_{p,q} = -\pi^2\left(\frac{p^2}{L_\xi^2}+\frac{q^2}{L_\eta^2}\right). \tag{3.12}$$

Now without loss of generality, we assume $L_\xi > L_\eta$, say $L_\eta = \mu L_\xi$ where $\mu \in (0,1)$. So the problem is to determine which eigenvector recovers the second important dimension and for what values of μ does it do this. I will present here a theoretical analysis of this problem and would confirm the theory numerically by looking at the eigenvectors of the diffusion map algorithm applied on the data set. Using (3.12), we have

$$\lambda_{0,0} = 0, \quad \lambda_{1,0} = -\pi^2\left(\frac{1}{L_\xi^2}\right), \quad \lambda_{2,0} = -\pi^2\left(\frac{4}{L_\xi^2}\right), \quad \lambda_{3,0} = -\pi^2\left(\frac{9}{L_\xi^2}\right),$$

$$\lambda_{4,0} = -\pi^2\left(\frac{16}{L_\xi^2}\right), \quad \lambda_{5,0} = -\pi^2\left(\frac{25}{L_\xi^2}\right), \quad \text{and} \quad \lambda_{0,1} = -\pi^2\left(\frac{1}{L_\eta^2}\right).$$

The first non-trivial eigenvector corresponds to $\lambda_{1,0}$, representing the most important coordinate since $L_\xi > L_\eta$ but then $\lambda_{0,1}$ corresponds to the second important coordinate. So the task is determining the location of $\lambda_{0,1}$ in terms of μ given that the eigenvalues are ordered in descending order.

First let us find for what values of μ, $\lambda_{0,1}$ comes second after $\lambda_{1,0}$ and before $\lambda_{2,0}$. That is

$$\lambda_{1,0} > \lambda_{0,1} > \lambda_{2,0} \tag{3.13}$$

which is equivalent to

$$\begin{aligned} -\pi^2\left(\frac{1}{L_\xi^2}\right) > -\pi^2\left(\frac{1}{L_\eta^2}\right) > -\pi^2\left(\frac{4}{L_\xi^2}\right) \\ \Leftrightarrow L_\xi^2 > L_\eta^2 > \frac{1}{4}L_\xi^2 \\ \Leftrightarrow L_\xi^2 > \mu^2 L_\xi^2 > \frac{1}{4}L_\xi^2 \\ \Leftrightarrow 1 > \mu^2 > \frac{1}{4} \qquad \text{thus} \quad \mu \in \left(\frac{1}{2}, 1\right). \end{aligned} \tag{3.14}$$

Taking a ratio of L_η to L_ξ of $\frac{3}{4}$ meaning $\mu = \frac{3}{4}$ which lies in the interval (3.14) I randomly generated data points as in Figure 3.6. I applied the diffusion map algorithm on these data points and plotted the non-trivial eigenvectors against ξ and η in the top and bottom rows of Figure 3.8 respectively. From Figure 3.8 we see clearly

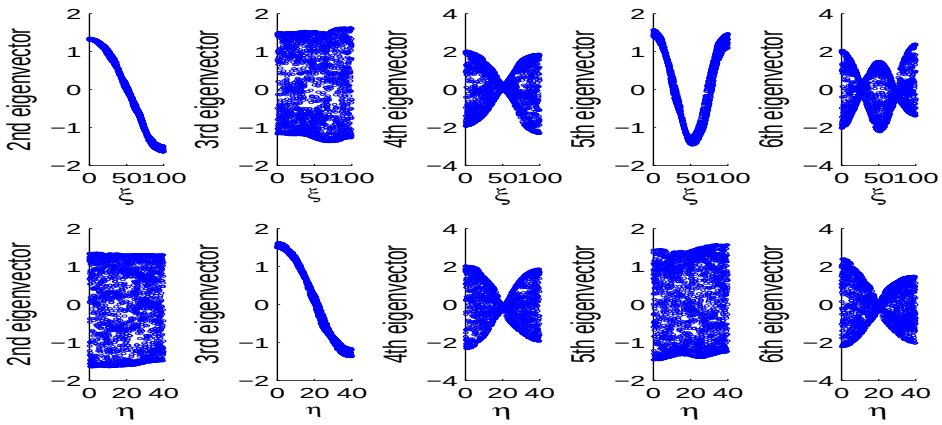


Figure 3.8: The eigenvectors against (top) ξ and (bottom) η with $\mu = \frac{3}{4}$.

that the first non-trivial eigenvector corresponds to $\lambda_{1,0}$ while the second non-trivial corresponds to $\lambda_{0,1}$ as expected from the above derivation.

On the other hand, for what values of μ does $\lambda_{0,1}$ come third after $\lambda_{1,0}$ and $\lambda_{2,0}$ but before $\lambda_{3,0}$? This occurs when the following inequality is satisfied:

$$\lambda_{2,0} > \lambda_{0,1} > \lambda_{3,0} \tag{3.15}$$

which is equivalent to

$$\begin{aligned} -\pi^2\left(\frac{4}{L_\xi^2}\right) > -\pi^2\left(\frac{1}{L_\eta^2}\right) > -\pi^2\left(\frac{9}{L_\xi^2}\right) \\ \Leftrightarrow \frac{1}{4}L_\xi^2 > L_\eta^2 > \frac{1}{9}L_\xi^2 \\ \Leftrightarrow \frac{1}{4}L_\xi^2 > \mu^2 L_\xi^2 > \frac{1}{9}L_\xi^2 \\ \Rightarrow \frac{1}{4} > \beta^2 > \frac{1}{9} \qquad \text{thus} \quad \mu \in \left(\frac{1}{3}, \frac{1}{2}\right). \end{aligned} \tag{3.16}$$

Again I chose μ value of 0.4 which lies in the interval 3.16. Then I generated the data points randomly on this domain and applied the diffusion map algorithm on those points. A plot of the eigenvectors versus the ξ and η is found in Figure 3.9. This time also we see the eigenvector corresponding to $\lambda_{0,1}$ coming third as expected from the above analysis.

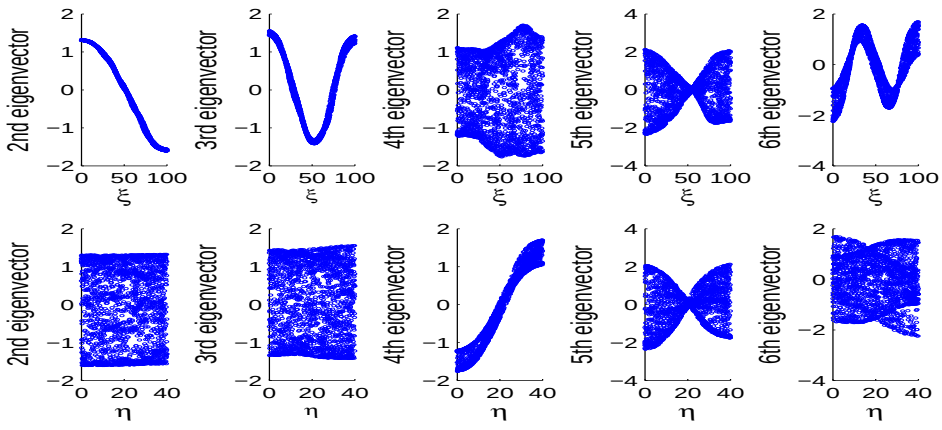


Figure 3.9: The eigenvectors against (top) ξ and (bottom) η with $\mu = 0.4$.

Similarly, for $\lambda_{0,1}$ to come fourth, fifth, sixth, etc $\mu \in (\frac{1}{4}, \frac{1}{3})$, $\mu \in (\frac{1}{5}, \frac{1}{4})$, $\mu \in (\frac{1}{6}, \frac{1}{5})$, etc respectively. However, numerically there exit a value of μ beyond which the second dimension seizes to be important and the embedded space becomes one dimensional in

effect. From my experience I am tempted to believe that and that value is very close to $\mu = \frac{1}{4}$. This is deducible from Figure 3.10 showing eigenvectors with $\mu = 0.225 < \frac{1}{4}$. In this figure none of the eigenvector plots against η gave a meaningful single sinusoidal mode as expected.

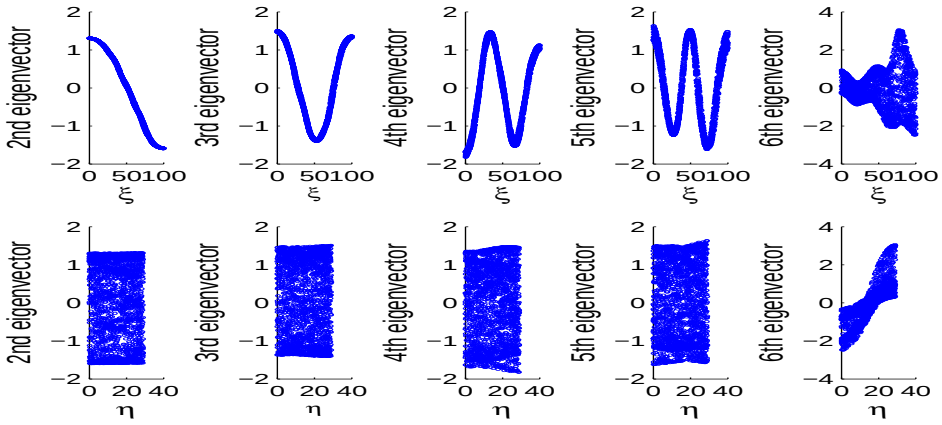


Figure 3.10: The eigenvectors against (top) ξ and (bottom) η with $\mu = 0.225$.

Chapter 4

SPARSE EIGENVALUE PROBLEMS

As afore-mentioned the core of the algorithm is solving Equation (1.3) $\mathbf{Ly} = \lambda\mathbf{Dy}$ which is a large and sparse generalised eigenvalue problem. In this Chapter, I discuss the techniques used to solve such problems, in particular the Implicitly Restarted Arnoldi Method (IRAM) which is the basis for the ARPACK routine `eigs` (`eig`) which I used in implementing the algorithm in Matlab [17].

Firstly we review some basic conventional notations which we will stick to for the rest of the chapter. $\mathbf{I}$ is the identity matrix in $\mathcal{C}^{k\times k}$. $\mathbf{e}_j$ is the j^{th} column of $\mathbf{I}$. $\mathbf{v}^T$ and $\mathbf{v}^H$ is the transpose of $\mathbf{v}$ and the complex conjugate of $\mathbf{v}^T$ respectively while $\mathbf{A}^T$, $\mathbf{A}^H$ and $\mathbf{A}^{-1}$ is the transpose, conjugate-transpose and the inverse of a matrix $\mathbf{A}$ respectively. $\det(\mathbf{A})$ is the determinant of $\mathbf{A}$ and $\mathcal{N}(\mathbf{A})$ is the nullspace of $\mathbf{A}$.

Given a matrix $\mathbf{A} \in \mathcal{C}^{k\times k}$ the pair $(\mathbf{v}, \lambda)$ that satisfy $\mathbf{Av} = \lambda\mathbf{v}$ are called an *eigenpair* where $\mathbf{v} \in \mathcal{C}^k$ is referred to as an *eigenvector* and $\lambda \in \mathcal{C}$ is referred to as an *eigenvalue*. I will look at the solution of a standard eigenvalue problem first then later in Section 2.5 extend it to a generalised eigenvalue problem. Algebraically, to get the eigenvalues we solve for the roots of the k^{th}-order *characteristic polynomial* $p(\lambda) = \det(\lambda\mathbf{I} - \mathbf{A})$ and the eigenvectors are given by any nonzero vector in the nullspace $\mathcal{N}(\lambda\mathbf{I} - \mathbf{A})$. However, we know from Galois theory [10, 20], for polynomials of degree greater than 4, its roots cannot be expressed as combinations of radicals of rational functions of the coefficients of the polynomial. Consequently we are forced to resort to iterative methods.

Generally speaking iterative methods for finding eigenvalues and eigenvectors are not in short supply. If we are interested in the leading eigenvalue of a simple $k \times k$ matrix the easiest would be to use the Power Method, see [20] for a description. If on the other hand it is several eigenvalues and eigenvectors of a dense problem we resort

to the QR Method discussed in the next section. If the problem is large and sparse we go for Arnoldi-based methods, like IRAM for large and sparse non-symmetric matrices or the Implicitly Restarted Lanczos Method (IRLM) for symmetric matrices.

The ARPACK routines are based on the IRAM as stated earlier. However, when the matrices involved a symmetric it reduces to the IRLM. The IRAM and IRLM can be considered as variants of the Arnoldi and Lanczos processes respectively combined with the Implicitly Shifted QR Method suitable for large scale problems. It would therefore be prudent to discuss briefly the Arnoldi method and the QR method before presenting the IRAM algorithm. The rest of the chapter would thus be discussing the QR method, Krylov subspaces and projection methods, the Arnoldi method, the IRAM and generalised eigenvalue problems, in the given order.

4.1 QR Method

The QR factorisation of $\mathbf{A} \in \mathcal{C}^{k\times k}$ is given by $\mathbf{A} = \mathbf{QR}$ where $\mathbf{Q} \in \mathcal{C}^{k\times l}$ is an orthogonal matrix and $\mathbf{R} \in \mathcal{C}^{l\times k}$ is an upper triangular matrix [29]. Basically, the QR method proceeds by performing a series of similarity transformations on the matrix $\mathbf{A}$ (the matrix whose eigenpair we a looking for) in the form $\mathbf{A}_{i+1} = \mathbf{Q}^T\mathbf{A}_i\mathbf{Q}$ such that the subdiagonal entries of $\mathbf{A}_i$ are made smaller at each iterative step until they are negligible [20]. This reduces $\mathbf{A}_i$ to a triangular matrix where its eigenvalues are given by the diagonal entries. $\mathbf{A}$ is usually first reduced to Hessenberg form $\mathbf{AV} = \mathbf{VH}$, where $\mathbf{V}^H\mathbf{V} = \mathbf{I}$ and $\mathbf{H} \in \mathcal{C}^{m\times m}$ is an upper Hessenberg matrix. An upper Hessenberg matrix is matrix in which the entries below the first subdiagonal are zero. We used this procedure because it can be very expensive to compute similarity transformations. This transformation can be easily achieved using Householder or Givens rotations [29, 27]. Below, in Table 4.1, is the QR method algorithm:

Table 4.1: **Algorithm 1:** QR method [20, 17].

Input: $(\mathbf{A}, \mathbf{V}, \mathbf{H})$ with $\mathbf{AV} = \mathbf{VH}, \mathbf{V}^H\mathbf{V} = \mathbf{I}$ and $\mathbf{H}$ is upper Hessenberg.
Output: $(\mathbf{V}, \mathbf{H})$ such that $\mathbf{AV} = \mathbf{VH}, \mathbf{V}^H\mathbf{V} = \mathbf{I}$ and $\mathbf{H}$ is upper triangular.

1. **for** $i = 1, 2, 3, \ldots$ until "convergence"
 1.1 Factor $\mathbf{H}_i = \mathbf{Q}_i\mathbf{R}_i$;
 1.2 Compute $\mathbf{H}_{i+1} = \mathbf{R}_i\mathbf{Q}_i$; and $\mathbf{V}_{i+1} = \mathbf{V}_i\mathbf{Q}_i$;
2. **end for**;

What is meant by convergence is when the subdiagonal entries are so small and less than a user specified tolerance.

Basically as the subdiagonal entries of $\mathbf{H}$ get closer to zero in magnitude the better its diagonal entries approximate the eigenvalues. Furthermore, the eigenvectors can be recovered from the accumulation of the product of the $\mathbf{Q}_i$ orthogonal matrices stored at each step. When we introduce a shift σ_i in the above algorithm we have the shifted QR method. The shift can be added explicitly to the diagonal of $\mathbf{H}_i$ during the implementation or otherwise. The former is the *explicitly shifted* QR method while the latter is the *implicitly shifted* QR method [20]. Table 4.2 presents the implicitly shifted QR method.

Table 4.2: **Algorithm 2:** Implicitly Shifted QR method [20, 17].

Input: $(\mathbf{A}, \mathbf{V}, \mathbf{H})$ with $\mathbf{AV} = \mathbf{VH}, \mathbf{V}^H\mathbf{V} = \mathbf{I}$ and $\mathbf{H}$ is upper Hessenberg.
Output: $(\mathbf{V}, \mathbf{H})$ such that $\mathbf{AV} = \mathbf{VH}, \mathbf{V}^H\mathbf{V} = \mathbf{I}$ and $\mathbf{H}$ is upper triangular.

1. **for** $i = 1, 2, 3, \ldots$ until "convergence"
 1.1 Select a shift σ_i;
 1.2 Factor $\mathbf{H}_i - \sigma_i\mathbf{I} = \mathbf{Q}_i\mathbf{R}_i$;
 1.3 Compute $\mathbf{H}_{i+1} = \mathbf{Q}_i^H\mathbf{H}_i\mathbf{Q}_i$; and $\mathbf{V}_{i+1} = \mathbf{V}_i\mathbf{Q}_i$;
2. **end for**;

The (m, m) element of $\mathbf{H}_i$ will converge to the eigenvalue closest to σ_i while the $(m, m-1)$ entry will converge to zero rapidly if the shift is a good estimate of an eigenvalue. The problem then can be deflated to a smaller problem once the convergence takes place. For practical purposes the (m, m) entry of $\mathbf{H}_i$ can be taken as the shift if the $(m, m-1)$ entry is small [20]. There are many other ways of improving on the convergence of the algorithm but a discussion on them would be too much of a digression, since this exposé is just a brief discussion on theme. It is important to note that `eig` command in Matlab is an efficient version of the QR method [20, 16, 17]. However, the QR method is not without limitations. Firstly, it destroys the sparsity and structure of the matrix. It also does not calculate all of the eigenvalues. It has storage implications when eigenvectors are desired. Thus the method is not suitable for large scale sparse matrices. Due to these drawbacks other methods have been formulated which include those discussed below.

4.2 Krylov Subspaces and Projection Methods

The main drawback of the power method (see a detail outline of the method in [20]) is that it computes a single dominant eigenvalue at a time and it overwrites the sequence of vectors which it produces during iteration. However, the set of the successive vectors known as the *Krylov subspace* contains some useful spectral information. The j^{th} Krylov subspace corresponding to $\mathbf{A}$ and the start vector $\mathbf{v}_0$ is given as:

$$\mathcal{K}_j(\mathbf{A}, \mathbf{v}_0) = Span\{\mathbf{v}_0, \mathbf{A}\mathbf{v}_0, \mathbf{A}^2\mathbf{v}_0, \ldots, \mathbf{A}^{j-1}\mathbf{v}_0\}.$$

Methods that attempt to extract information from this space are called *Krylov subspace* or *projection* methods and the underlying algorithms of the ARPACK subroutines are derived from these class of methods [20, 16, 17].

The underlying principle is to construct approximate eigenvectors in the Krylov subspace $\mathcal{K}_j(\mathbf{A}, \mathbf{v}_0)$ by imposing a Galerkin condition. A vector $\mathbf{u} \in \mathcal{K}_j(\mathbf{A}, \mathbf{v}_0)$ is called a *Ritz vector* with a corresponding *Ritz value* θ if the pair $(\mathbf{u}, \theta)$ satisfy the *Galerkin condition*

$$\mathbf{v}^T(\mathbf{A}\mathbf{u} - \lambda\mathbf{u}) = 0 \qquad \text{for all } \mathbf{v} \in \mathcal{K}_j(\mathbf{A}, \mathbf{v}_0).$$

This means that the Ritz pair are equivalent to an eigenpair in the projection onto a smaller space. The smaller the orthogonal component to the space the better the approximation to the eigenpair of $\mathbf{A}$. This is what the condition hope to achieve [20].

4.3 Arnoldi factorisations

The Arnoldi method can be seen as a powerful extension of the power method and can be used to find j eigenpairs of a matrix simultaneously. Simply put this method is a way of generating an orthonormal basis $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j\}$ for $\mathcal{K}_j(\mathbf{A}, \mathbf{v}_0)$ [29]. A *j-step Arnoldi factorisation* of $\mathbf{A} \in \mathcal{C}^{k\times k}$ is given by the following relation:

$$\mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{H} + \mathbf{f}\mathbf{e}_j^T$$

where the columns of $\mathbf{V} \in \mathcal{C}^{k\times m}$ are orthonormal, $\mathbf{f}$ is the residual vector, $\mathbf{V}^H\mathbf{f} = \mathbf{0}$ and $\mathbf{H} \in \mathcal{C}^{m\times m}$ as afore-stated is upper Hessenberg. It should be noted that the above relationship becomes a *j-step Lanczos factorisation* if $\mathbf{A}$ is Hermitian (symmetric) and $\mathbf{H}$ would then be real, symmetric and tridiagonal [20, 17].

If $\mathbf{H}\mathbf{u} - \theta\mathbf{u} = \mathbf{0}$ then $\mathbf{v} = \mathbf{V}\mathbf{u}$ satisfies

$$\|\mathbf{A}\mathbf{v} - \theta\mathbf{v}\| = \|(\mathbf{A}\mathbf{V} - \mathbf{V}\mathbf{H})\mathbf{u}\| = \gamma|\mathbf{e}_j^T\mathbf{u}|$$

where $\gamma = \|\mathbf{f}\|$ and $\gamma|\mathbf{e}_j^T\mathbf{u}|$ is called the Ritz estimate of the Ritz pair $(\mathbf{v}, \lambda)$. The basis for the Arnoldi factorisation is to approximate the eigenpair for $\mathbf{A}$ from the eigenpair for $\mathbf{H}$, which is usually a far smaller matrix and its eigenpair can be computed by conventional (dense) methods. The target is to reduce the Ritz estimate to zero. At least this happens when $\mathbf{f} = \mathbf{0}$ in which case $\mathbf{V}$ will be an invariant subspace of $\mathbf{A}$ and the eigenpair of $\mathbf{A}$ will correspond exactly to the Ritz pair [20]. A $(j+1)$-step and a $(j+p)$-step Arnoldi factorisation is given in Tables 4.3 and 4.4 respectively.

Table 4.3: **Algorithm 3:** $(j+1)$-step Arnoldi factorisation [20, 17].

Input: $(\mathbf{V}_j, \mathbf{H}_j, \mathbf{f}_j)$ with $\mathbf{A}\mathbf{V}_j = \mathbf{V}_j\mathbf{H}_j + \mathbf{f}_j\mathbf{e}_j^T$.
Output: $(\mathbf{V}_{j+1}, \mathbf{H}_{j+1}, \mathbf{f}_{j+1})$ such that $\mathbf{A}\mathbf{V}_{j+1} = \mathbf{V}_{j+1}\mathbf{H}_{j+1} + \mathbf{f}_{j+1}\mathbf{e}_{j+1}^T$.

1. Compute $\gamma_j = \|\mathbf{f}_j\|$ and $\mathbf{v} = \mathbf{f}_j/\gamma_j$;
2. Construct $\mathbf{V}_{j+1} = (\mathbf{V}_j, \mathbf{v})$ and $\mathbf{H}_{j+1} = \begin{pmatrix} \mathbf{H}_j \\ \gamma_j\mathbf{e}_j^T \end{pmatrix}$;
3. Compute $\mathbf{z} = \mathbf{A}\mathbf{v}_{j+1}$;
4. Compute $\mathbf{h}_{j+1} = \mathbf{V}_{j+1}^T\mathbf{z}$ and $\mathbf{f}_{j+1} = \mathbf{z} - \mathbf{V}_{j+1}\mathbf{h}_{j+1}$;
5. Construct $\mathbf{H}_{j+1} = (\mathbf{H}_{j+1}, \mathbf{h}_{j+1})$;
6. **end**;

Table 4.4: **Algorithm 4:** $(j+p)$-step Arnoldi factorisation [20, 17].

Input: $(\mathbf{V}_j, \mathbf{H}_j, \mathbf{f}_j)$ with $\mathbf{A}\mathbf{V}_j = \mathbf{V}_j\mathbf{H}_j + \mathbf{f}_j\mathbf{e}_j^T$.
Output: $(\mathbf{V}_{j+p}, \mathbf{H}_{j+p}, \mathbf{f}_{j+p})$ such that $\mathbf{A}\mathbf{V}_{j+p} = \mathbf{V}_{j+p}\mathbf{H}_{j+p} + \mathbf{f}_{j+p}\mathbf{e}_{j+p}^T$.

1. **for** $i = j, j+1, \ldots, j+p-1$
   1.1. Compute $\gamma_i = \|\mathbf{f}_i\|$ and $\mathbf{v} = \mathbf{f}_i/\gamma_i$;
   1.2. Construct $\mathbf{V}_{i+1} = (\mathbf{V}_i, \mathbf{v})$ and $\mathbf{H}_{i+1} = \begin{pmatrix} \mathbf{H}_i \\ \gamma_i\mathbf{e}_i^T \end{pmatrix}$;
   1.3. Compute $\mathbf{z} = \mathbf{A}\mathbf{v}_{i+1}$;
   1.4. Compute $\mathbf{h}_{i+1} = \mathbf{V}_{i+1}^T\mathbf{z}$ and $\mathbf{f}_{i+1} = \mathbf{z} - \mathbf{V}_{i+1}\mathbf{h}_{i+1}$;
   1.5. Construct $\mathbf{H}_{i+1} = (\mathbf{H}_{i+1}, \mathbf{h}_{i+1})$;
2. **end for**;

Reorthogonalisation would have to be done in order for the columns of $\mathbf{V}$ to remain orthogonal during numerical implementation of the Arnoldi factorisation. This is achieved by Gram-Schmidt process of some steps of iterative refinement. However,

usually we do not need all the steps only one step is enough, [20]. To implement this we replace step 1.5 in Algorithm 4 with the following also from [20]:

1.5 **for** $\nu = 1, 2, \ldots$ until "convergence"

1.5.1 $\mathbf{s} = \mathbf{V}_{i+1}^T \mathbf{f}_{i+1}$;

1.5.2 $\mathbf{f}_{i+1} = \mathbf{f}_{i+1} - \mathbf{V}_{i+1}\mathbf{s}$;

1.5.3 $\mathbf{h}_{i+1} = \mathbf{h}_{i+1} + \mathbf{s}$;

1.6 $\mathbf{H}_{i+1} \leftarrow (\mathbf{H}_{i+1}, \mathbf{h}_{i+1})$;

4.4 Implicitly Restarted Arnoldi Method

It is worth noting that the Arnoldi factorisation is totally dependent on the choice of a starting vector $\mathbf{v}_0$. Depending on the application it is generally desired that $\mathbf{v}_0$ be rich in the subspace spanned by the wanted eigenvectors with very little component in the direction of unwanted eigenvectors. There will be need to restart the Arnoldi factorisation after an adaptive refinement of $\mathbf{v}_0$ to be a linear combination of approximations to our desired eigenvectors, proposed by Saad [22] based on the polynomial acceleration scheme developed by Manteuffel [26] for the iterative solution of linear systems [17]. Based on the implicitly shifted QR factorisation, the implicitly restarted Arnoldi method could accomplish this without any explicit computation of a new Arnoldi factorisation, [20].

The basic principle of the method is to extend the j-step Arnoldi factorisation

$$\mathbf{A}\mathbf{V}_j = \mathbf{V}_j\mathbf{H}_j + \mathbf{f}_j\mathbf{e}_j^T$$

to a $(j+p)$-step Arnoldi factorisation

$$\mathbf{A}\mathbf{V}_{j+p} = \mathbf{V}_{j+p}\mathbf{H}_{j+p} + \mathbf{f}_{j+p}\mathbf{e}_{j+p}^T$$

and apply p implicit shifts to this factorisation to obtain

$$\mathbf{A}\mathbf{V}_+ = \mathbf{V}_+\mathbf{H}_+ + \mathbf{f}_{j+p}\mathbf{e}_{j+p}^T$$

where $\mathbf{V}_+ = \mathbf{V}_{j+p}\mathbf{Q} = \mathbf{Q}^H\mathbf{H}_{j+p}\mathbf{Q}$ and $\mathbf{Q} = \mathbf{Q}_1\mathbf{Q}_2\ldots\mathbf{Q}_p$, where $\mathbf{Q}_i$ is related to factoring $(\mathbf{H} - \sigma_i\mathbf{I}) = \mathbf{Q}_i\mathbf{R}_i$. We assume that this factoring can be done easily since $\mathbf{H}$ is small. It could be seen that $\mathbf{V}_+$ being a product of $\mathbf{V}$ and an orthogonal matrix $\mathbf{Q}$ has orthonormal columns. It turns out also $\mathbf{H}_+$ is upper Hessenberg [20]. Thus the shift σ does not disturb the structure of the Arnoldi factorisation. Interestingly enough, the first $(j-1)$ elements of $\mathbf{e}_{j+p}\mathbf{Q}$ are zero, thus equating the first j columns on each side will give a new j-step Arnoldi factorisation

$$\mathbf{A}\mathbf{V}_j^+ = \mathbf{V}_j^+\mathbf{H}_j^+ + \mathbf{f}_j^+\mathbf{e}_j^T.$$

This new j-step factorisation can then be extended to a $(j+p)$-step factorisation, applying shifts, and condensing. This is equivalent to applying a p^{th} degree polynomial in $\mathbf{A}$ to $\mathbf{v}_0$ and the roots of this polynomial correspond to the p shifts applied. So the trick is to choose the shifts to be the "unwanted" eigenvalues so that we enhance the components of $\mathbf{v}_0$ in the direction of the desired eigenvectors [20]. The Exact Shift Strategy discussed in [20] could be one of the ways to select the shifts efficiently. Table 4.5 presents the implicitly restarted Arnoldi method.

Table 4.5: **Algorithm 5:** Implicitly Restarted Arnoldi method [20, 17].

Input: Matrix $\mathbf{A}$ whose eigenpair are sought, the number of eigenpair sought j, the number of implicit shifts p to apply to the factorisation at each step, the sort criterion S for deciding desired eigenvalues, start vector $\mathbf{v}_0$ and a tolerance τ

Output: $\{(\mathbf{v}_1, \lambda_1), (\mathbf{v}_2, \lambda_2), \ldots, (\mathbf{v}_j, \lambda_j)\}$, approximations to the j eigenpair of $\mathbf{A}$.

1. Using v_0 as a starting vector, generate a j-step Arnoldi factorisation
   $\mathbf{AV} = \mathbf{VH} + \mathbf{fe}_j^T$.
2. **for** $i = 1, 2, \ldots$ until $\|\mathbf{Av}_i - \lambda_i \mathbf{v}_i\| < \tau$ for all $i = 1, 2, \ldots, j$
   2.1. Extend the j-step Arnoldi factorisation to a $(j+p)$-step Arnoldi factorisation
   $\mathbf{AV} = \mathbf{VH} + \mathbf{fe}_{j+p}^T$.
   2.2. Let $\mathbf{q} = \mathbf{e}_{j+p}$
   2.3. Sort the eigenvalues of $\mathbf{H}$ from best to worst according to the sort criterion S and take $\{\sigma_1, \ldots, \sigma_p\}$ to be the p worst eigenvalues.
   2.4. **for** $\nu = 1, 2, \ldots, p$
      2.4.1. Factor $\mathbf{H} - \sigma_\nu \mathbf{I} = \mathbf{QR}$.
      2.4.2. Compute $\mathbf{H} = \mathbf{Q}^H \mathbf{HQ}$.
      2.4.3. Compute $\mathbf{V} = \mathbf{VQ}$.
      2.4.4. Compute $\mathbf{q} = \mathbf{q}^H \mathbf{Q}$.
   2.5. Compute $\mathbf{f} = \mathbf{V}_{j+1} \times \mathbf{H}_{j+1} + \mathbf{f} \times \mathbf{q}_j$.
   2.6. Take the first j columns on each side of the factorisation to get
   $\mathbf{V} = \mathbf{V}_j, \quad \mathbf{H} = \mathbf{H}_j$.
   2.7. Take as eigenpair approximations $(\mathbf{v}_i, \lambda_i)$ as Ritz pairs of the problem.
3. **end for**;

4.5 Generalised eigenvalue problems

Generalised eigenvalue problems of the form $\mathbf{Ly} = \lambda \mathbf{Dy}$ can also be solved using the methods discussed above. As $\mathbf{D}$ is symmetric and nonsingular, the generalised

problem can then be easily transformed into a standard eigenvalue problem, since

$$\mathbf{Ly} = \lambda \mathbf{Dy}$$

is equivalent to

$$\mathbf{D}^{-1}\mathbf{Ly} = \lambda \mathbf{y}.$$

With this standard formulation we apply Arnoldi on $\mathbf{D}^{-1}\mathbf{L}$. To avoid loss of symmetry, structure and sparsity, the matrix $\mathbf{D}^{-1}\mathbf{L}$ is not formed but instead we compute $\mathbf{w} = \mathbf{D}^{-1}\mathbf{Ly}$ in the following two steps [20]:

1. Solve $\mathbf{z} = \mathbf{Ly}$;
2. Compute $\mathbf{Dw} = \mathbf{z}$.

Another way we can transform our generalised problem to a standard form without loosing the symmetry, structure and sparsity of $\mathbf{D}$ is to do the following transformation.

$$\mathbf{Ly} = \lambda \mathbf{Dy}$$
$$\mathbf{D}^{-1}[\mathbf{D} - \mathbf{W}]\mathbf{y} = \lambda \mathbf{y}$$
$$[\mathbf{I} - \mathbf{D}^{-1}\mathbf{W}]\mathbf{y} = \lambda \mathbf{y}.$$

Let $\mathbf{K} = \mathbf{D}^{-1}\mathbf{W}$. Thus finding the eigenvectors corresponding to the smallest eigenvalues of our generalised problem is the same as doing the same thing for $\mathbf{I} - \mathbf{K}$, which in turn is equivalent to finding the largest eigenvalues of $\mathbf{K}$. Since $\mathbf{L}$ is positive semidefinite all the eigenvalues of $\mathbf{I} - \mathbf{K}$ are non-negative. Therefore all the eigenvalues of $\mathbf{K}$ are smaller or equal to 1. Then we proceed to compute the eigenpair of $\mathbf{K}$ by using the fact that

$$\mathbf{K} = \mathbf{D}^{-1/2}\mathbf{S}\mathbf{D}^{1/2} \quad \text{where } \mathbf{S} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$$

is a symmetric matrix. The symmetry of $\mathbf{S}$ implies it is diagonalisable with k eigenvalues and eigenvectors, $\mathbf{U}_i, i = 1, \ldots, k$ that form an orthonormal basis of $\mathbb{R}^k$ which means the eigenvectors of $\mathbf{K}$ are then given by $\mathbf{V}_i = \mathbf{D}^{-1/2}\mathbf{U}_i$ [8].

On the other hand, we can factor $\mathbf{D}$ by $\mathbf{D} = \mathbf{C}\mathbf{C}^T$ using the Cholesky factorisation where $\mathbf{C}$ is a lower triangular matrix. Note, I changed the conventional $\mathbf{L}\mathbf{L}^T$ Cholesky factorisation notation to $\mathbf{C}\mathbf{C}^T$ in order to avoid confusion with the $\mathbf{L}$ in our generalised problem. The factorisation can be done if $\mathbf{D}$ is positive definite. Then again we transform the problem into standard form as $\mathbf{C}^{-1}\mathbf{L}\mathbf{C}^{-T}\mathbf{w} = \lambda \mathbf{w}$ where $\mathbf{w} = \mathbf{C}^T\mathbf{y}$. It should be remarked that the eigenvectors of the generalised problem are

not orthogonal but orthogonal with respect to $\mathbf{D}$ *inner product* $(\mathbf{y}, \mathbf{w})_D = \mathbf{w}^H\mathbf{D}\mathbf{y}$; which means given any two eigenvectors, say $\mathbf{y}$ and $\mathbf{w}$, of the matrix pencil $(\mathbf{L}, \mathbf{D})$, then $\mathbf{w}^H\mathbf{D}\mathbf{y} = 0$. This is justified by the fact that the matrix $\mathbf{D}^{-1}\mathbf{L}$ is self-adjoint with respect to the $\mathbf{D}$ inner product [20, 15].

Shift and invert techniques can be used if we are interested in the eigenvalues of the generalised problem closest to a certain shift σ. In this case the matrix of interest can be determine as thus:

$$\begin{aligned} \mathbf{L}\mathbf{y} &= \lambda\mathbf{D}\mathbf{y} \\ (\mathbf{L} - \sigma\mathbf{D})\mathbf{y} &= \mathbf{L}\mathbf{y} - \sigma\mathbf{D}\mathbf{y} = (\lambda - \sigma)\mathbf{D}\mathbf{y} \\ \mathbf{D}^{-1}(\mathbf{L} - \sigma\mathbf{D})\mathbf{y} &= (\lambda - \sigma)\mathbf{y} \\ (\mathbf{L} - \sigma\mathbf{D})^{-1}\mathbf{D}\mathbf{y} &= \frac{1}{(\lambda - \sigma)}\mathbf{y}. \end{aligned}$$

After this transformation we can now resort to the Arnoldi method to compute eigenvalues of the matrix $(\mathbf{L} - \sigma\mathbf{D})^{-1}\mathbf{D}$ and doing another standard transformation to recover the desired eigenvalues of the original generalised problem. Like before the matrix $(\mathbf{L} - \sigma\mathbf{D})^{-1}\mathbf{D}$ is never formed but we proceed by computing the matrix-vector product $\mathbf{w} = (\mathbf{L} - \sigma\mathbf{D})^{-1}\mathbf{D}\mathbf{y}$ as given in [20]:

1. Factor $(\mathbf{L} - \sigma\mathbf{D}) = \mathbf{E}\mathbf{F}$;
2. Solve $\mathbf{z} = \mathbf{D}\mathbf{y}$;
3. Compute $\mathbf{E}\mathbf{d} = \mathbf{z}$;
4. Compute $\mathbf{F}\mathbf{y} = \mathbf{d}$.

The matrix pair $\mathbf{E}\mathbf{F}$ are the $\mathbf{L}\mathbf{U}$ in the LU factorisation. This notation change was necessary again to avoid confusion between $\mathbf{L}$ in the generalised problem and $\mathbf{L}$ in the LU factorisation. However, the LU factorisation can be performed taking advantage of the sparsity of $\mathbf{L}$ and $\mathbf{D}$. Moreover, it needs to be done only once, while step 3 and 4 can be accomplish quickly using backwards substitution [20].

Chapter 5

APPLICATIONS OF DIFFUSION MAPS

I would now present two little applications of diffusion maps. The first is clustering, discussed in Section 5.1. Specifically I used K-means to cluster some test data and to perform colour reduction in colour quantization. The second is low dimensional description of high dimensional sets of images discussed in Section 5.2.

5.1 Clustering

In my comment on the features of diffusion maps in Chapter 1, I mentioned the relationship of the diffusion map algorithm to clustering. Clustering which is popularly used in many fields including machine learning, data mining, pattern recognition, image analysis and bio-informatics is the unsupervised classification or partitioning of data set into classes (clusters) where class members have some similarity [9]. In this chapter I will show that the diffusion map algorithm can be used to perform clustering. In doing so I compare it to a highly used clustering technique, K-means clustering in Section 5.1.1. Then I would further show that like K-means, diffusion maps could be used for Vector Quantization in Section 5.1.2.

5.1.1 Ordinary clustering

There are many types of clustering and many difference methods of clustering. Here I will look at Partitional clustering and in particular the K-means clustering method in comparison with diffusion maps used for clustering. The K-means algorithm assigns each point to the cluster whose centroid (centre) is nearest. The centroid is the average of all the points in the cluster [9]. The algorithm is composed of the following steps (J. MacQueen, 1967):

1. Decide the number of clusters K, and select K points as the initial centroids.

2. Assign each point to the nearest centroid.

3. Re-compute the coordinates of the centroids.

4. Repeat steps 2 and 3 until the centroids no longer change position.

The K-mean algorithm is simple and relatively fast, thus can be used on large data sets. However, its main drawback is that each different initial assignment of centroids converges to a different set of clusters.

I randomly generated some clusters. So I have a data set of predefined clusters on which I would apply both K-means and diffusion maps to recover these clusters. In Figure 5.1 I show these clusters of the original data set. In the left panel I plot three clearly defined clusters and on the right four clusters.

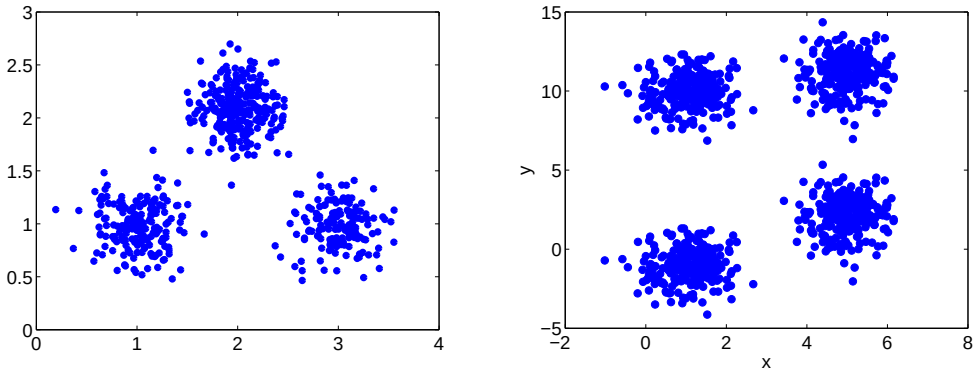


Figure 5.1: A plot of the 3 and 4 clusters I generated respectively.

The left panel of Figure 5.2 shows one of the results from K-means with random initial centroids. The black squares represent the centroids. The right panel shows the result from the application of diffusion maps colouring the points according the second eigenvector of the map. The key thing for the diffusion map is the careful choice of ε in the heat kernel. Based on the criterion for the choice of ε in Section 3.2, I used $\varepsilon = 0.12$. Both plots show that both algorithm were able to recover the three clusters.

Similarly, I applied the K-mean algorithm with random initial starting centroids on the data set with four clusters. Then to the same data I applied the diffusion map algorithm with $\varepsilon = 0.78$. The results are shown in left and right panels of Figure 5.3 respectively. Again both algorithms recovered the four clusters.

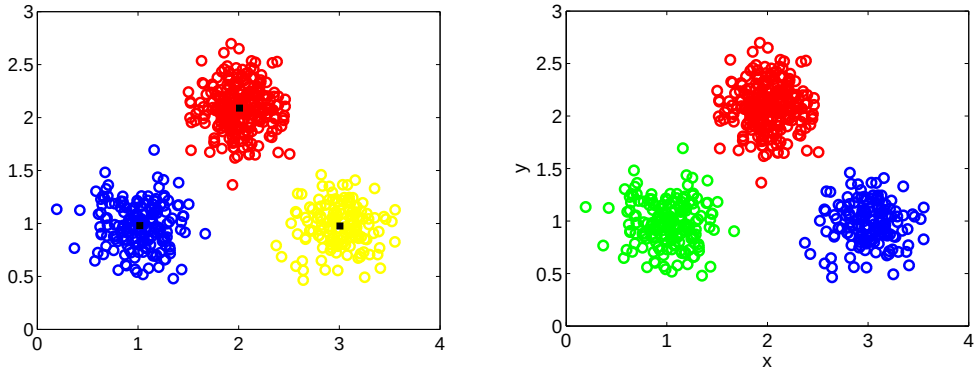


Figure 5.2: Recovery of the 3 clusters using K-mean clusters (left) and using the first three eigenvectors of the diffusion map (right).

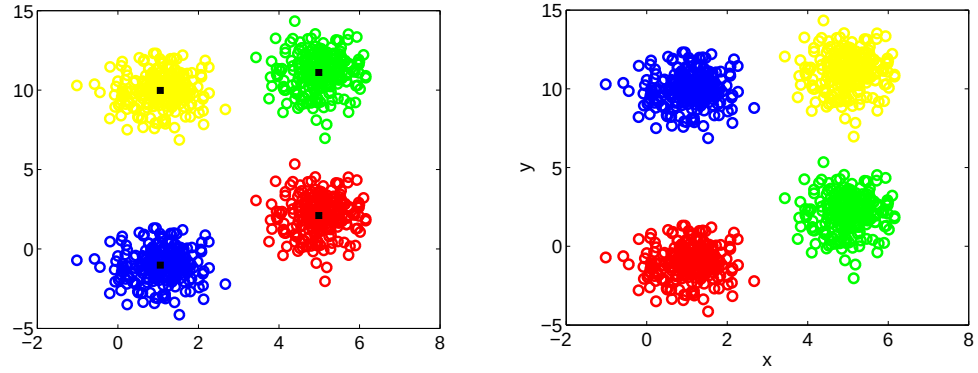


Figure 5.3: Recovery of the 4 clusters using K-mean clusters (left) and using the first three eigenvectors of the diffusion map (right).

So diffusion maps could do exactly what K-means does. In fact with an added advantage over K-means which is there is no need to predetermine the number of clusters. However, the choice of ε can be non trivial especially if the data set is large. It should be noted that the ε is proportional to the diffusion time. The larger the ε the more diffusion takes place and the reverse is also true. So at the appropriate time determined by the appropriate choice of ε we could recover all the clusters. Beyond such a time or with a larger ε the diffusion map may not find the appropriate clusters, see [3].

It is even more interesting to discover that diffusion maps succeed were K-means fails. Here is an example of two circular clusters one inside the other. In the left panel of Figure 5.4, we have two clusters on inside the other. The result of using K-means to recover the clusters is shown in the middle panel. While the result of using diffusion maps is shown in the right panel. It is the diffusion map that was

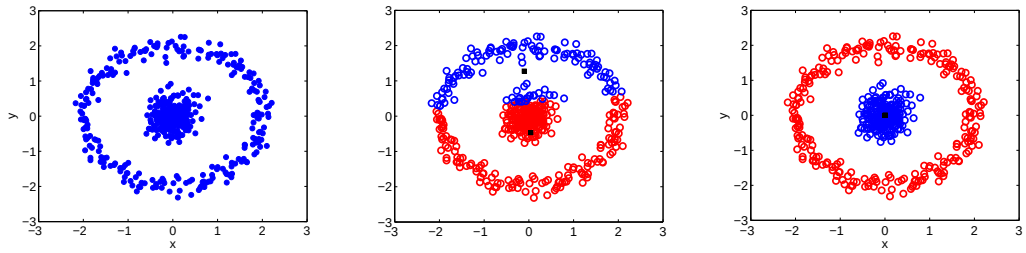


Figure 5.4: Circular clusters, (left) original clusters, (middle) clustering by K-means and (right) clustering by diffusion maps.

able to identify the clusters the way I wanted them to be seen. There is no doubt therefore that diffusion maps are a very useful clustering algorithm [9].

5.1.2 Colour quantization

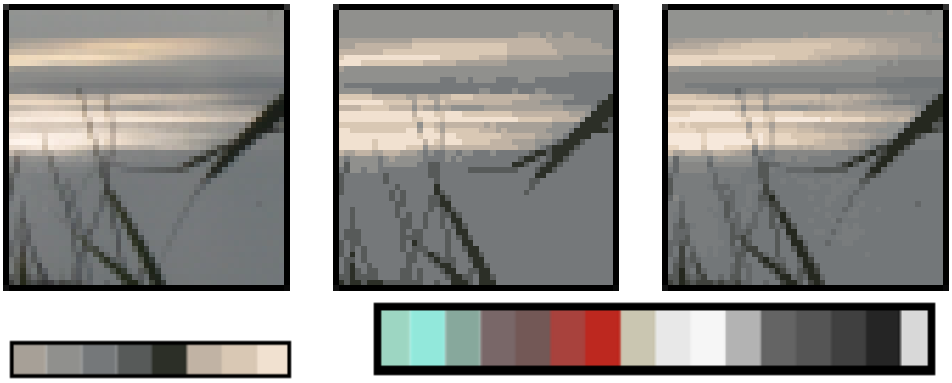


Figure 5.5: Sample image one, (top left) original image (size 50×50 pixels), K-means colour reduced images of $8-$bits (top middle) and $16-$bits (right). Bottom images are the $8-$bit and $16-$bit RGB palettes respectively.

Colour quantization or colour image quantization is a form of image compression. It is used to reduce the number of distinct colours used in an image so that images with many colours could be displayed on devices that can only display a limited number of colours, usually due to memory limitations [12, 9]. Colours are represented as three-dimensional vectors of floating point values. Each component represents the amount of Red, Green and Blue in the RGB colour scheme or one of the components of any other colour scheme used.

It works by clustering these sets of vectors (points) into clusters and representing each cluster by its centroid (also called *codevector*), as in K-means and some other clustering algorithms. The set of codevectors known as the *codebook* form the colour

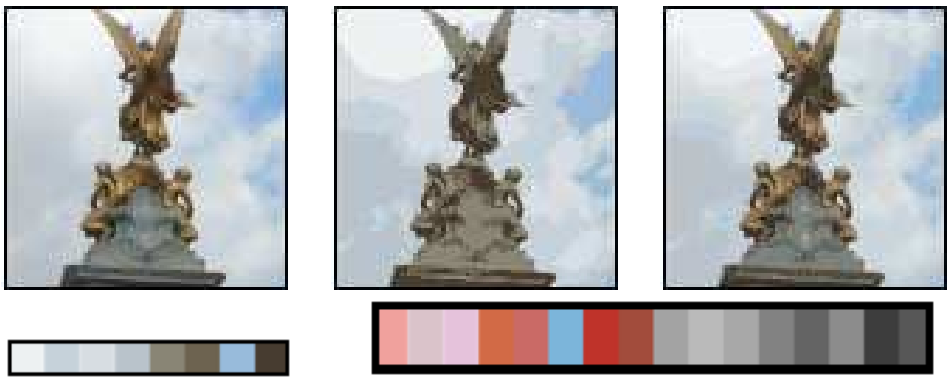


Figure 5.6: Sample image one, (top left) original image (size 100 × 100 pixels), K-means colour reduced images of 8−bits (top middle) and 16−bits (right). Bottom images are the 8−bit and 16−bit RGB palettes respectively.

palette of the new image. Almost any three-dimensional clustering algorithm can be applied to colour quantization, including diffusion maps [12, 9].

Using the RGB scheme I implemented the classical K-means on a sample of three images of different colour variation and different sizes. The results are shown in Figure 5.5, Figure 5.6, and Figure 5.7.

Figure 5.7: Sample image one, (top left) original image (size 200 × 200 pixels), K-means colour reduced images of 8−bits (top middle) and 16-bits (right). Bottom images are the 8−bit and 16−bit RGB palettes respectively.

Visually, the 16−bit colour quantization is much better and looks closer to the original than the 8−bit. Similarly, diffusion maps are expected to produce similar results theoretically. Basically what the diffusion map does is to map the data set into a feature space or in other words stretches the data into a 1D manifold by the first non-trivial eigenvector which then can be cluster by observing thresholds or even K-means.

To demonstrate this I used the original image in Figure 5.5 and map its colour data points by the second eigenvector of the algorithm. Then I used K-means to cluster this eigenvector into 8 and 16 clusters respectively and clustered the data points according to the clustering of this eigenvector. From which I calculated the code vectors and created the colour palette of the reduced colour image. The original and the reduced colour $8-bit$ and $16-bit$ images are shown in Figure 5.8 with the colour palettes of the 8−bit and 16−bit images respectively.

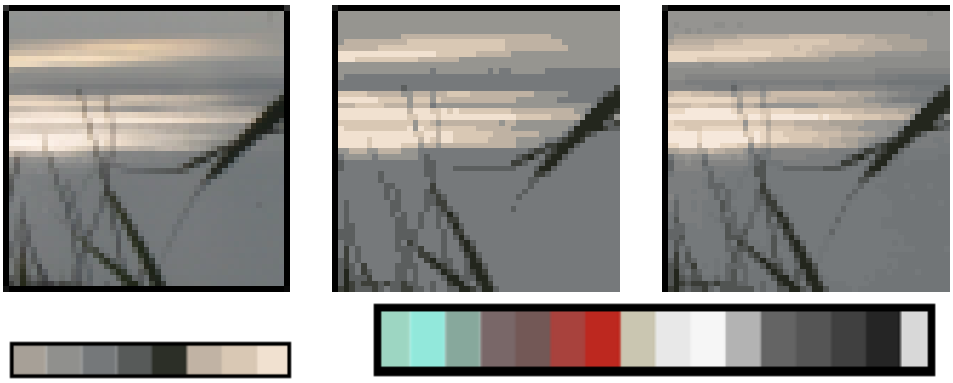


Figure 5.8: Sample image one, (top left) original image (size 50×50 pixels), diffusion map colour reduced images of 8−bits (top middle) and 16−bits (right). Bottom images are the 8−bit and 16−bit RGB palettes respectively.

As we can see from Figures 5.5 and 5.8, the diffusion colour reduction produce almost exactly the results as the K-means colour reduction. The only problem is the processing time for the diffusion map algorithm. While it takes ordinary K-means under a minute to process the image in Figure 5.5, it takes the diffusion map algorithm more than 5 minutes to process the same image shown in Figure 5.8. For the image in Figure 5.6 which took K-means a little over a minute, the diffusion map took more than 10 minutes. I did not show the result of that because it is identical to the K-means results in Figure 5.6.

The simple reason for this discrepancy in processing time between the two algorithms is that the number of clusters K is pre-determined in K-means unlike the diffusion maps which have to calculate mutual distances between each point in the data set. So considering the processing time I would not recommend the use of diffusion maps for colour quantization. In fact there are improved K-mean algorithm that claim to produce much better results and are relatively faster to implement than ordinary K-means, see [7, 12, 2].

5.2 Low dimensional representation of images

Now I would show the capability of the diffusion map algorithm in recovering low dimensional representation of high dimensional data set of images, see original motivation in Figure 1.1. This can also be seen as a reorganisation or a reordering capability of the algorithm. To do this I took 32 pictures of my faces rotated horizontally about the direction am facing taken by a fixed camera. These pictures are cropped to 50×50 pixels for easy processing in Matlab.

Figure 5.9: Pictures in the original file ordered from left to right and top to bottom.

Originally the pictures are indexed by the angle of the head as can be seen in Figure 5.9. Then I mixed the order pictures randomly (see Figure 5.10) as I inputted them into the algorithm. In the implementation of the algorithm each image is treated a point in $\mathbb{R}^{50\times 50}$. As the algorithm preserves mutual distances it is able to exactly discover the order of the pictures which can be considered a low dimensional representation of the high dimensional input image space. Ordering the pictures according to the second eigenvector of the algorithm gave an image exactly identical to Figure 5.9, that is why I decided not to show it separately.

However, to further confirm this agreement I plotted the index of the order in which I mixed the images and the index from the order of the second eigenvector in the left panel of Figure 5.11. We can see that the two curves agree 100%. Shown in an alternative way, the two orderings are plot against each other in the right panel of Figure 5.11 and the mapping is one-to-one as expected.

Figure 5.10: Pictures re-ordered randomly as they were inputted into the implementation of the algorithm.

Furthermore, this agrees with results obtained by Lafon in [13] and Tenenbaum et al. in [28]. However, Tenenbaum et al. were using LLE and this goes further to show the closeness or connection between LLE and diffusion maps. So I agree with [13, 28] that this technique could yields satisfactory results when applied to discrimination of images.

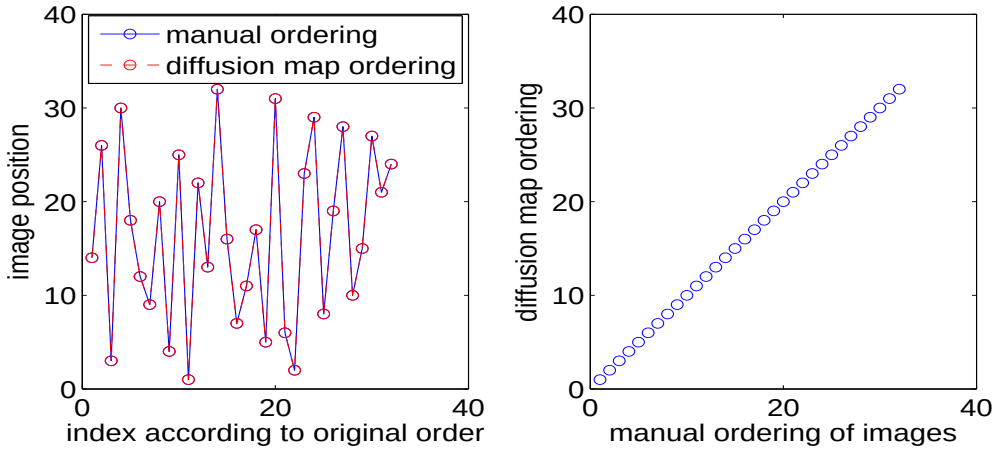


Figure 5.11: Pre-implementation ordering and ordering according to the second eigenvector (left) and pre-implementation ordering against ordering according to the second eigenvector (right).

Chapter 6

CONCLUSIONS

In summary what this dissertation tried to do is to show *what* the diffusion map algorithm is; *how* it works; *why* it works; and *where* it works with a little digression discussing solution of sparse matrix problems which is key to the implementation of the algorithm.

Chapter 1 being the introduction was presented with a style. Instead of boring the reader with literature about the algorithm it first tried to put the whole method into context as a dimensionality reduction method. Then I went straight ahead to present the algorithm and two simple illustrative examples that convincingly show that the method works. The chapter concluded with a brief literature review and a description of the key features of the algorithm. So from this chapter it is clear that dimensionality reduction is a problem we are confronted with in almost every field of research and that the diffusion map technique, which is relatively new but has quite a number of paper written on it ([19, 8, 4, 25] just a few out of the lot), is one of the approaches to use to tackle the dimensionality reduction problem. In presenting the algorithm I just stuck to one version as there are a family of them, [19]. The simple examples confirmed that the algorithm works as papers on the subject showed with slightly more sophisticated examples. The brief literature review showed that the method is related to many other dimensionality reduction methods like eigenmaps and LLE which might be even considered variants of the diffusion map method [4]. The particularly striking key feature of the algorithm out of a host of other features is its close connection to spectral clustering which has useful application in biological perception and machine vision [1].

Chapter 2 attempts to answer the question: why does the algorithm work? It justifies the method theoretically by showing that it is equivalent to the minimisation of some objective functions which is a mapping that preserves locality information. It also showed that minimising this objective function was equivalent to minimising

(2.3) and that the vector that minimises (2.3) is the eigenvector corresponding to the smallest eigenvalue of the generalised eigenvalue problem (1.3). A little but necessary digression at the end of this chapter discussed the choice of the scaling parameter ε in the algorithm. This could be viewed as a disadvantage of the method, since at anytime you are dealing with a new data set you need to determine what is an appropriate ε to use and this could be time consuming.

In a further attempt to justify the method. I decided to reinterpret the algorithm as some form of a random walk on a graph in Chapter 3. I showed that the random walk on the graph is equivalent to the probabilistic heat equation on the domain described by the graph with a Neumann boundary condition, see the derivation of Equations (3.3) and (3.4). This claim was substantiated numerically by the agreement between the results of some random walk averaged over the number of runs and the solution of the probabilistic heat equation on the same domain.

Either random walk or the solution of the heat equation can be quite complicated and difficult to accomplish. However, they could be approximated by the first few eigenvectors of the Laplacian on the domain which is in effect the diffusion map algorithm. Finally, the last section of the chapter investigated the behaviour of the eigenvectors of the Laplacian on a 2D domain embedded in a higher dimensional domain. I discovered particular intervals which tell when will the second dimension be captured by the eigenvectors of the Laplacian and this agreed perfectly with the results from the application of the diffusion map algorithm on the data set in the domain.

Chapter 4 was a brief literature review of solutions of sparse matrix problems. This was particularly important to me because I wanted to understand the numerical linear algebra behind the `eigs` command I used to implement the algorithm [17] and I thought there is no better way than to write about it. So there are methods out there to solve eigenvalue problems but the most suitable for sparse problems presently is the Implicitly Restarted Arnoldi method which is implemented by the `eigs` ARPACK routine called by Matlab. IRAM becomes IRLM if the matrices involved are symmetric and the two methods are actually a synthesis of the Arnoldi/Lanczos methods with the implicitly restarted QR method.

In the final chapter, Chapter 5, I looked at applications of the algorithm. This is to find out where the algorithm works. Given the connection between spectral clustering and the diffusion map method mentioned earlier I thought it would be interesting to explore this relationship. The examples in the ordinary clustering shows that the diffusion map method is as good as K-means with the advantage of not needing to

specify the number of clusters in advance like K-means do. On the other hand the size of ε has to be appropriate otherwise the algorithm will fail to capture the exact number of clusters. Reference [3] called it the sampling time. If it is too small or too large the algorithm merges or divides some clusters. However, we clearly see the algorithm outperforming the K-means where the data set is nonlinear and the dimension very high.

Colour quantization was the other clustering application I tested the algorithm on. Here again there were encouraging results except that the processing time for diffusion maps was for longer than K-means, moreover other colour quantization algorithms claimed to be much faster and give better quality than K-means. Thus, it would not be a recommended technique for colour quantization given the time it takes to come up with similar or even lower quality results as compared to other techniques. Finally, the last section of Chapter 5 demonstrated the low dimensional description of a set of images. This low dimensional representation was basically the ordering of the images. The results presented here showed the algorithm is very good for such tasks and this agrees with results presented by Lafon in his thesis [13]. This has far reaching applications in image processing and computer vision.

On the overall this dissertation has not invented or created anything new apart from the behaviour of the eigenvectors discussed in Chapter 3 which was missing from the literature of diffusion maps. However, it was able to confirm results of earlier works on this subject using simple examples. I must comment that most of the data sets I used were test examples generated by me. This is to make it more convincing given that I know what results I expect. However, I have no doubt that the methods I used will perfectly work on any real life data. An obvious continuation of this work is to use real life data.

Appendix 1

Principal component analysis (PCA)

PCA has been could be referred to as one of the most valuable results from applied linear algebra. It is a simple, non-parametric method of extracting relevant information from confusing data sets. PCA provides a means of reducing a complex data set to a lower dimension to reveal the sometimes hidden, simplified structure that often underlie it particularly if the data set linear [24].
Basically performing PCA includes the following step:

1. Organize a data set as an $m \times n$ matrix, where m is the number of measurement types and n is the number of trials.

2. Subtract off the mean for each measurement type.

3. Calculate the eigenvectors of the covariance.

Below is a Matlab Code for the implementation of PCA from [24]:
This code is written for Matlab 6.5 (Release 13) from Mathworks13. The code is not computationally efficient but explanatory (terse comments begin with a %).

```
function [signals,PC,V] = pca1(data)
% PCA1: Perform PCA using covariance.
% data - MxN matrix of input data
% (M dimensions, N trials)
% signals - MxN matrix of projected data
% PC - each column is a PC
% V - Mx1 matrix of variances

[M,N] = size(data);
% subtract off the mean for each dimension
```

```
mn = mean(data,2);
data = data - repmat(mn,1,N);

% calculate the covariance matrix
covariance = 1 / (N-1) * data * data;

% find the eigenvectors and eigenvalues
[PC, V] = eig(covariance);

% extract diagonal of matrix as vector
V = diag(V);

% sort the variances in decreasing order
[junk, rindices] = sort(-1*V);
V = V(rindices);
PC = PC(:,rindices);

% project the original data set
signals = PC * data;
```

Appendix 2

Table 6.1: A Matlab pseudo code of the diffusion map algorithm [8].

Input: $\mathbf{X} \in \mathbb{R}^{k\times n}$ the set of data points, $\mathbf{v} = (1, 1, \ldots, 1)^T \in \mathbb{R}^k$ and $\varepsilon \in \mathbb{R}$.
Output: $\mathbf{V} \in \mathbb{R}^{k\times m}$ the set of normalised eigenvectors, $\mathbf{\Lambda} \in \mathbb{R}^{k\times m}$ the diagonal matrix with the eigenvalues as the diagonal entries arranged in descending order.

1. **for** $i = 1, 2, 3, \ldots, k$
 1.1 Compute $\mathbf{W}$ by $W_{i,j} = \exp\left(\frac{\|x_i - x_j\|^2}{\varepsilon}\right)$;
2. **end for**;
3. Form the matrix $\mathbf{D} = \texttt{diag}(\mathbf{W} \times \mathbf{v})$;
4. Compute $\mathbf{K} = \mathbf{D}^{-1}\mathbf{W}$;
5. Compute $\mathbf{S} = \mathbf{D}^{(-0.5)} \times \mathbf{W} \times \mathbf{D}^{(-0.5)}$;
6. Compute $\mathbf{U}$ and $\mathbf{\Lambda}$ by $[\mathbf{U}, \mathbf{\Lambda}] = \texttt{eigs}(\mathbf{S})$;
7. Compute $\mathbf{V} = \mathbf{D}^{(-0.5)} \times \mathbf{U}$;
8. Compute $\mathbf{V} = \mathbf{V}/\mathbf{V}_{1,1}$;

References

[1] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, **6**(15):1373–1396, June 2003.

[2] C. Chang and Y. Hu. A fast lbg codebook training algorithm for vector quantization. *IEEE Transactions on Consumer Electronics*, **44**(4):1201–1208, November 1998.

[3] R. Coifman and S. Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, **21**:5–30, 2006.

[4] R. Coifman et al. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the National Academy of Sciences*, **102**(21):7426–7431, May 2005.

[5] J. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[6] D. Donoho and C. Grimes. Hessian eigenmaps locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, **100**(10):5591–5596, May 2003.

[7] W. Equitz. A new vector quantization clustering algorithm. *IEEE Transactions on Acoustics, Speech and Signal Processing*, **37**(10), October 1989.

[8] R. Erban et al. Variable-free exploration of stochastic models: A gene regulatory network example. *The Journal of Chemical Physics*, **126**(15):155103, April 2007.

[9] M. Filippone et al. A survey of kernel and spectral methods for clustering. *Pattern Recognition*, **41**:176–190, January 2008.

[10] I. Herstien. *Topics in Algebra*. John Wiley and Sons, 1975.

[11] I. Kevrekidis. Personal communication. 2008.

[12] K. Krishna et al. Vector quantization using genetic k-means algorithm for image compression. *Proceedings of 1997 International Conference on Information, Communications and Signal Processing*, **3**:1585–1587, September 1997.

[13] S. Lafon. *Diffusion maps and geometric harmonics*. PhD thesis, Yale University, U.S.A, May 2004.

[14] S. Lafon and A. Lee. Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and data set parameterization. *IEEE Transactions on Pattern analysis and machine intelligence*, **28**(9):1393–1403, September 2006.

[15] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, **45**(4):255–282, October 1950.

[16] R. Lehoucq and D. Sorensen. Deflation techniques for an implicitly restarted arnoldi iteration. *SIAM Journal Matrix Analysis and Application.*, **17**(4):789–821, October 1996.

[17] R. Lehoucq et al. Arpack users guide: Solution of large scale eigenvalue problems with implicitly restarted arnoldi methods. Technical report, 1998.

[18] B. Mohar and S. Poljak. *Eigenvalues in Combinatorial Optimisation*, **50** of *Combinatorial and Graph-Theoretical Problems in Linear Algebra*. Springer-Verlag, 1993.

[19] B. Nadler et al. Diffusion maps, spectral clustering and reaction coordinates of dynamical systems. Submitted to Applied and Computational Harmonic Analysis; cite as arXiv:math/0503445v1 [math.NA], March 2005.

[20] R. Radke. *A Matlab implementation of the implicitly restarted Arnoldi method for solving large-scale eigenvalue problems*. Master's thesis, Rice University, Hoston, Texas, April 1996.

[21] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, **290**(5500):2323–2326, December 2000.

[22] Y. Saad. *Numerical Methods for Large Eigenvalue Problems.* Manchester University Press, 1992.

[23] S. Seung and D. Lee. The manifold ways of perception. *Science*, **290**(5500):2268–2269, December 2000. Perspectives: Cognition.

[24] J. Shlens. A tutorial on principal component analysis. e-mail: shlens@salk.edu, December 2005.

[25] A. Singer et al. Detecting the slow manifold by anisotropic diffusion maps. Submitted to The Proceedings of the National Academy of Sciences of the USA, 2007.

[26] D. Sorenson. Implicitly restarted arnoldi / lanczos methods for large scale eigenvalue calculations. In et al. Keyes, D., editor, *Parallel Numerical Algorithms*, pages 23–25, Hampton, May 1995. ICASE/LaRC Workshop, VA Kluwer.

[27] G. Stewart. *Introduction to Matrix Computations.* Academic Press, 1973.

[28] J. Tenenbaum et al. A global geometric framework for nonlinear dimensionality reduction. *Science*, **290**(5500):2319–2323, December 2000.

[29] A. Wathen. Numerical linear algebra. lecture notes, 2007.

[30] Z. Zhang and H. Zhang. Principal manifolds and nonlinear dimension reduction via local tangent space alignment. Cse-02-019, Pennsylvania State University, Department of computer science and engineering, 2002.