Greedy Principal Flows

National University of Singapore

Sebastian Lie

05 March 2021

Acknowledgements

My 7 soft toys. And my desk lamp for being the light of my life.

Abstract

Principal Flows are a great tool to use when we want to extend the notion of Principal Component analysis to multivariate datasets that we know lie on non-linear manifolds. We restrict this problem to constructing principal flows on hyperspheres. We use a different, easier method to obtain the principal flow that is even closer to its canonical PCA interpretation.

Contents

Introduction

0.1 Motivation

For principal flows or motivation for coding algorithm?

With the advent of Big Data, using machine learning on multivariate datasets to solve problems from disparate fields has grown in popularity. Unsupervised learning has also grown in popularity, Principal Component Analysis being the most popular algorithm. However, if it is given data that lie on some manifold, or a non-linear space, PCA fails to achieve good results. This is why we want to use Principal flows: in essence they are a generalisation of PCA to manifolds. Furthermore, in keeping with the trends of the data science field, we want to use a popular language, python to construct these principal flows.

0.2 Background and Definitions

We first define notation and some important concepts we will use later. We will work throughout with a

Notation	**Explanation**
$\mathbb{R}^D$	D dimensional space of real numbers.
X	Data Matrix, D dimensions
D	The original dimension of the data matrix
d	Dimension of lower dimensional data which X is reduced to
M	Denote a Manifold embedded in $\mathbb{R}^d$
p	Denotes a point on the manifold, M.
v	Denotes a vector
T_pM	Denotes the Tangent space of a point p on the Manifold.
S	Denotes a Covariance matrix
{x_1,...x_n}	Denotes a collection of points on the manifold, M.

0.2.1 Definitions

Now we define the notion of vector fields.

0.2.2 Vector Fields

Contextualise vector fields in terms of the eigenvector field: $\mathbf{X}(p) = (p, \epsilon(p))$.

A vector *at point p*, $p \in \mathbb{R}^{n+1}$ is a pair $\mathbf{v} = (\mathbf{p}, \mathbf{v}),\ v \in \mathbb{R}^{n+1}$, such that $\mathbf{v}$ is the vector v translated so that its tail is at p instead of the origin. All vector operations are defined such that the first item of the pair remains the same, and the second item is the result of the operation. The length and angle between 2 vectors are the same as normal vectors rooted in the origin.

Definition: A *vector field* $\mathbf{X}$ on $U \subset \mathbb{R}^{n+1}$ is a function which assigns to each point of U a vector at that point.

$$\mathbf{X}(p) = (p, X(p))$$

for some function $X : U \longrightarrow \mathbb{R}^{n+1}$. Vector fields on $\mathbb{R}^{n+1}$ are often most easily described by specifying this associated function X.

Take for example, the eigenvector field: $\mathbf{X}(p) = (p, \epsilon(p))$

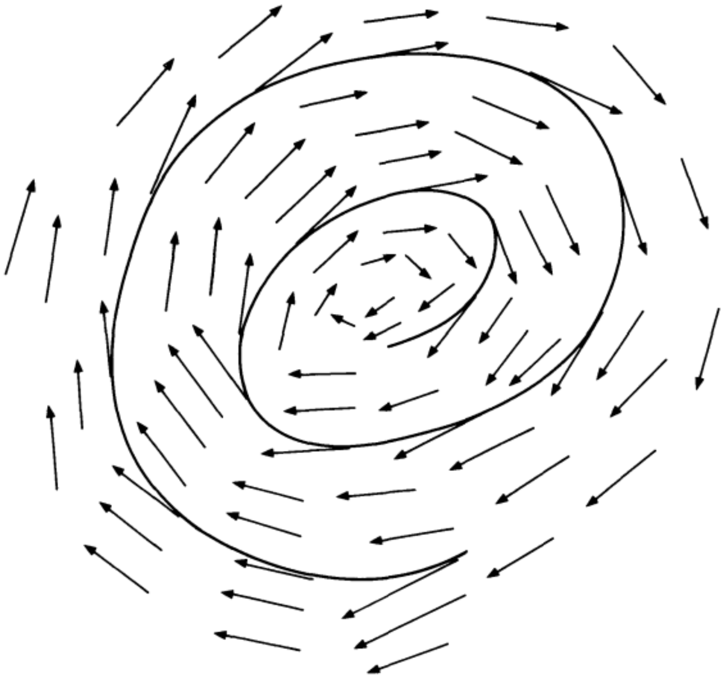


Figure 2.5 An integral curve of a vector field.

Figure 1: An example of a vector field, $\mathbb{R}^3$

0.2.3 Graphs and Level Sets

Definition: Given a function $f : U \longleftarrow \mathbb{R}$, where $U \subset \mathbb{R}^{n+1}$. it's level sets $f^{-1}(c)$ are defined, for each real number c, by

$$f^{-1}(c) = \{(x_1, ..., x_{n+1}) \in U : f(x_1, ...x_{n+1}) = c\}$$

i.e all the input values for f that result in a value of c.

Then c is known as the *height* of the level set and $f^{-1}(c)$ is the level set at height c, or the solution set of the equation $f(x_1, ...x_{n+1}) = c$.

Definition: The *graph* of a function $f : U \longleftarrow \mathbb{R}$, is the subset of $\mathbb{R}^{n+2}$ defined by

$$graph(f) = \{(x_1, ...x_{n+2}) \in \mathbb{R}^{n+2} : (x_1, ...x_{n+1}) \in U \ and \ x_{n+2} = f(x_1, ...x_{n+1})\}$$

For $c \geq 0$ the level set of f at height c is just the set of all points in the domain of f over which the graph is at distance c. Graphs are simply the illustration of the function where one of the axes is the output of f itself, and the other axes are its inputs. Fixing the x_{n+2} value of the graph gives a level set, as illustrated below.

For n = 1, level sets are (at least for non-constant differentiable functions) generally curves in $\mathbb{R}^2$. These curves play the same roles as **contour lines** on a topographic map. Just as contour maps provide an accurate picture of the topography of a land, so does a knowledge of the level sets and their heights accurately portray the graph of a function.

Often it is hard to **visualise a graph** in higher dimensions, and thus we use level sets to help us do so. The level set of a graph of f with 3 dimensional input can be visualised in this way: let x_i = constant, and the plane formed will cut the level set $f^{-1}(c)$ (c fixed) in some subset, usually a curve. Letting the plane move, by changing the selected value of the xi-coordinate, these subsets will generate the level set $f^{-1}(c)$. This is illustrated below.

0.2.4 Logarithm Maps

Logarithm Map: For each $p \in M$, let

$$log_p(x) : M \longrightarrow T_pM$$

be the logarithm map. The log map is a function that projects a point x_i on the manifold M onto the plane tangent to M at p, by producing a vector which indicates the direction in which p should move to obtain the projection of $x \in M$ onto T_pM.

Exponential Map: For each $p \in M$, let

$$exp_p(x) : T_pM \longrightarrow M$$

be the exponential map. Exponential maps are the inverse of the logarithm maps. Given a point p, the exponential map w.r.t M, a manifold, is a function that projects a point from the plane tangent to M at p onto x on the manifold M.

0.2.5 Tangent Space

Contextualise tangent space as the space at T_pM.

> For a smooth function f, a vector u is said to be *tangent to the level set* $f^{-1}(c)$ if it is a velocity vector of a parametrized curve in $\mathbb{R}^{n+1}$ whose image is contained in $f^{-1}(c)$.

0.2.6 Geodesics

Geodesics extend the concept of straight lines in the Euclidean space to Manifolds. A geodesic is the straight line distance between x_i and x_j on some manifold M.

0.2.7 Eigenvalues and Eigenvectors

Now we define eigenvalues and eigenvectors **Definition**: Let $\mathbf{A} \in \mathbb{R}^{n \times n}$. Then a non-zero vector $\mathbf{v}$ is an eigenvector of $\mathbf{A}$ if there exists some scalar λ such that $\mathbf{Av} = \lambda \mathbf{v}$. Then λ is known as the eigenvalue corresponding to vector $\mathbf{v}$.

Here we also note that for any 2 eigenvectors v_i and v_j, $v_i \cdot v_j = 0$, or any 2 eigenvectors are orthogonal to each other.

0.2.8 Diagonalisation

Literature Review

should be 4 pages long

0.3 Linear Dimension Reduction

Linear Dimensionality Reduction[1]**:** *Given n D-dimensional data points:* $\boldsymbol{X} = [x_1, ...x_n]^T \in \mathbb{R}^{n \times D}$ *and a choice of dimensionality* $d < D$*, optimise some objective function* $f_X(\dot{})$ *to produce a linear transformation* $P \in \mathbb{R}^{p \times r}$ *and call* $Y = XP \in \mathbb{R}^{n \times d}$

0.3.1 Principal Component Analysis

Principal Component analysis is a linear dimension reduction method that tries to obtain a lower-dimensional representation of the data that retains as much variation as possible present in the data set. PCA does this by constructing principal components (PCs) which are linear combinations of the original variables. These PCs are uncorrelated (orthogonal) and ordered in descending order by the amount of variation retained. Then reducing some high-dimensional data of dimension D to a lower dimensional d is simply a matter of computing the first d principal components.

Algorithm

- 1. First we center the data matrix, **X**, by

- 2. Next we compute the covariance matrix of the centered data matrix.

[1] This definition from here.

- 3. Now we diagonalise the covariance matrix and take the first d eigenvectors. Let these d eigenvectors be the columns of a matrix $\mathbf{V}_d$.

- 4. Multiply $\mathbf{X}$ by $\mathbf{V}_d$ to obtain the d principal components.

0.3.2 Classical MDS, Euclidean Distance

Multidimensional Scaling is a linear dimensionality reduction method. Its main aim is to take some high dimensional data, $\mathbf{X}$ and find some low dimensional points so as to minimize the discrepancy between the pairwise distances in the original space and the pairwise distances in the lower-dimensional space.

Let a squared dissimilarity matrix between n points be $\mathbf{M} \in \mathbb{R}^{n \times n}$, let the matrix of coordinates be denoted by $\mathbf{X} \in \mathbb{R}^{n \times D}$, and let $\mathbf{B} = \mathbf{X}\mathbf{X}'$. Since dissimilarities do not change under translations, we assume that $\mathbf{X}$ has column means equal to 0. MDS seeks to find a lower dimensional representation $\mathbf{X}_{(d)} \in \mathbb{R}^{n \times d}$.

Algorithm: from 2005 Book Modern Multidimensional Scaling, pg 262

- 1. Compute or obtain the dissimilarity matrix, $\mathbf{D}$.

- 2. Let $\mathbf{J}$ be the centering matrix: $\mathbf{J} = \mathbf{I} - n^{-1}\mathbf{1}\mathbf{1}'$. Compute $\mathbf{B} = -\frac{1}{2}\mathbf{J}\mathbf{D}\mathbf{J}$.

- 3. Then, compute the eigendecomposition of $\mathbf{B}$: $\mathbf{B} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}'$.

- 4. Then $\mathbf{X} = \mathbf{\Lambda}^{1/2}\mathbf{V}'$ and for a q-dimensional representation of $\mathbf{X}$: $\mathbf{X}_{(p)}$, $\mathbf{X}_{(p)} = \mathbf{\Lambda}_p^{1/2}\mathbf{V'}_p$, where $\mathbf{\Lambda}_p^{1/2}$ is the first $p \times p$ submatrix of Λ, and $\mathbf{V'}_p$ is the first p columns of $\mathbf{V}$', i.e the first q eigenvectors and their corresponding eigenvalues.

Note that using Euclidean distances, the result of MDS is the same as PCA. Advantages of MDS over PCA: If we have $n << p$, then MDS is likely to be more efficient, since MDS finds eigenvectors of a $n \times n$ matrix and a $p \times p$ matrix respectively. (From 2002 book on Principal Component Analysis) MDS simple to implement, and their optimizations do not involve local minima despite their inherent limitations as linear methods.

0.4 Non-Linear Dimension Reduction

Non-linear dimensionality reduction methods are particularly useful when the multivariate data we obtain is sampled from a smooth non-linear manifold, e.g a manifold in an S-shape, obtains better estimates than linear methods like PCA and MDS, which implicitly assume that the data is sampled from a linear space. Certain data sets contain essential nonlinear structures that are invisible to PCA and MDS.

0.4.1 Principal Geodesic Analysis

Principal Geodesic Analysis(PGA) is a generalization of principal component analysis to connected, complete manifolds. The main aim of PGA is to be able to describe the variability of some data lying on some manifold M. More formally, the goal is to find a sequence of lower-dimensional subspaces, which on manifolds are nested geodesic submanifolds that maximize the projected variance of the data. These submanifolds are called the principal geodesic submanifolds, which are analogous to linear subspaces for PCA.

Let T_pM be the tangent space of M at the intrinsic mean p of the x_i. The intrinsic mean of some data $\mathbf{X}$ lying on some manifold is obtained by first setting the initial mean to a random data point, then iteratively obtaining a better estimation of the intrinsic mean by computing the average of the vectors obtained using the log map at the current mean on all data points, then taking the next estimate of the intrinsic mean as the projection of that average using the exponential map at the current estimate.

Let $U \subset T_pM$ be a neighbourhood of 0 in which our data, $\{x_1, ...x_n\}$ lies, such that projection is well defined for all geodesic submanifolds of $exp_p(U)$.These principal geodesic submanifolds are obtained by by first constructing an orthonormal basis of tangent vectors $v_1, ..., v_d \in T_pM$ that span the tangent space. These vectors are then used to form the principal geodesic subspaces H_k where $H_k = exp_p(V_k)$, and V_k is intersection of the subspace spanned by vectors $1..k$ and U. subspace $V_k = span(\{v_1, ..., v_k\}) \bigcap U$.

Algorithm:

- 1. Obtain the intrinsic mean, $p \in M$ of $\{x_1, ..x_n\}$.
- 2. Calculate the vectors $u_i = log_p(x_i)$.
- 3. Calculate the covariance matrix $\mathbf{S} = \frac{1}{n}\sum_{i=1}^{n} u_i u_i^T$.
- 4. Diagonalise the covariance matrix to obtain $\{v_k, \lambda_k\}$ the eigenvectors and eigenvalues

respectively, which represent the principal directions in the tangent space T_pM and the variances.

0.4.2 Locally Linear Embedding

Locally Linear Embedding

The main idea of LLE is to construct a neighbourhood preserving mapping of the original points of dimension D to the reconstructed points of dimension d. We assume that for some data $\mathbf{X} \in \mathbb{R}^D$, the data lie on or near a smooth manifold of dimensionality $d << D$. Locally, LLE assumes the embedding is linear, and thus for each data point $p \in \mathbb{R}^D$, we aim to use a linear combination of its K nearest neighbours to reconstruct a lower-dimensional $p \in \mathbb{R}^d$. LLE does this by first learning some reconstruction weights from the D-dimensional data: $\mathbf{W}$ where $\mathbf{W}_{ij}$ represents the contribution of the j-th data point in reconstructing the i-th one. These weights obey an important symmetry: for any particular data point, they are invariant to rotations, rescalings, and translations of that data point and its neighbors. They thus reflect intrinsic geometric properties of the It then uses the same weights $\mathbf{W}_{ij}$ that reconstruct the ith data point in D dimensions to also reconstruct its embedded manifold coordinates in d dimensions. (Should I explain why LLE does this?) At the end of LLE, each D-dimensional observation $\mathbf{X}_i$ is mapped to a low dimensional vector $\mathbf{Y}_i$ representing global internal coordinates on the manifold.

Embedding Cost Function:

$$\Phi(\mathbf{Y}) = \sum_i |\mathbf{Y}_i - \sum_j \mathbf{W}_{ij}\mathbf{Y}_j|^2$$

LLE Algorithm:

- 1. Compute the K nearest Neighbors of each original data point $\mathbf{X}_i$, where K is a hyperparameter.
- 2. Compute the weights $\mathbf{W}_{ij}$ that best reconstruct each data point from it's neighbours

minimizing the **Reconstruction Error** below by constrained linear fits.

$$\varepsilon(\mathbf{W}) = \sum_i |\mathbf{X}_i - \sum_j \mathbf{W}_{ij}\mathbf{X}_j|^2$$

- 3. Compute the vectors $\mathbf{Y}_i$ best reconstructed by the weights $\mathbf{W}_{ij}$ by minimising the quadratic form in eq (2) by its bottom d non-zero eigenvectors.

Isomap

Isomap is an approach that combines the major algorithmic features of PCA and MDS — computational efficiency, global optimality, and asymptotic convergence guarantees—with the flexibility to learn a broad class of nonlinear manifolds. It can be thought of as an extension of MDS to manifolds. Isomap achieves this by estimating the geodesic distance between data points, given only input-space distances, e.g euclidean distance between points.

This relies on the fact that for neighboring points, input-space distance provides a good approximation to geodesic distance. For faraway points, Isomap approximates geodesic distance by adding up a sequence of "short hops" between neighboring points, computed efficiently by finding shortest paths in a graph with edges connecting neighboring data points. This approximation relies on the proof that for a sufficiently high density (a) of data points, we can always choose a neighborhood size (e or K) large enough that the graph will (with high probability) have a path not much longer than the true geodesic, but small enough to prevent edges that "short circuit" the true geometry of the manifold.

Isomap Algorithm:

- 1. Construct neighbourhood graph G: First, we need to compute the distances between points: for any node i and j, connect the 2 nodes if $d(i,j) < \epsilon$ or if j is one of the k-nearest neighbours of i.

- 2. Compute all-pairs shortest paths on G. There are many algorithms to do this, but we use the Floyd-Warshall algorithm.

- 3. Construct d-dimensional embedding, using MDS.

Greedy Principal Flows

0.5 Problem Statement

The main objective for greedy principal flows was to quantify or describe multivariate data on the manifold: we cannot simply fit a line, as this is not Euclidean space. Instead, we want some curve or path such that, locally, it follows the path of maximal variation of the data. This is done by taking the largest eigenvector from the diagonalisation of the covariance matrix. This can be thought of as the manifold extension of the euclidean 1st principal component.

Problem statement:

0.6 An explanation of the algorithm

Algorithm explanation: Assume for now that our data, $\mathbf{X}$ lies on the hypersphere. We know that the first principal component passes through the centroid of the data. We want our principal flow to do the same. Thus, we first need to find the centroid of the data. Start at an arbitrary data point, p. Then we project all other data points onto T_pM, then calculate the covariance matrix of the points on T_pM and find the first eigenvector from diagonalisation, and move in that direction.

Then we start from that centroid, and build our principal flow from there.

0.7 Algorithm Steps

Defn: The Principal Flow of the dataset is basically an integral curve that is always tangent to the direction of 'maximal variation' at any given point it contains.

- Computes the principal flow of the dataset. Idea: This is a "greedy" implementation of the principal flow algorithm, developed originally by Professor Yao Zhi Gang.

- We assume the underlying structure of the data is a hypersphere. Starting from the centroid of the data set (user defined or calculated below), we apply the following procedure:

- 1. Project the data residing on the hypersphere onto the hyperplane tangent to the centroid of the data (p). We use the log map of p, obtaining a matrix of vectors on the tangent plane that point from p to the projected points. These are the plane vectors.

- 2. Compute the largest principal component of the points using the plane vectors, applying weights as necessary via the kernel function provided.

- 3. The largest principal component is the new directions that the principal flow moves in. We determine it's sign by making sure it is moving in the same direction as the previous principal direction.

- 4. We take a small step in each direction on the plane, then project it back to the hypersphere.

- 5. We do 1-4 for the point on the opposite end of the growing principal flow.

- 6. Store both points.

- 7. Repeat 1-6 until max_iter is reached.

Principal Boundary

- Computes the principal flow of the dataset. Idea: This is a ”greedy” implmentation of the principal flow algorithm, developed originally by Professor Yao Zhi Gang.
- We assume the underlying structure of the data is a hypersphere. Starting from the centroid of the data set (user defined or calculated below), we apply the following procedure:
- 1. Project the data residing on the hypersphere onto the hyperplane tangent to the centroid of the data (p). We use the log map of p, obtaining a matrix of vectors on the tangent plane that point from p to the projected points. These are the plane vectors.
- 2. Compute the largest principal component of the points using the plane vectors, applying weights as necessary via the kernel function provided.
- 3. The largest principal component is the new directions that the principal flow moves in. We determine it’s sign by making sure it is moving in the same direction as the previous principal direction.
- 4. We take a small step in each direction on the plane, then project it back to the hypersphere.
- 5. We do 1-4 for the point on the opposite end of the growing principal flow.
- 6. Store both points.
- 7. Repeat 1-6 until max_iter is reached.

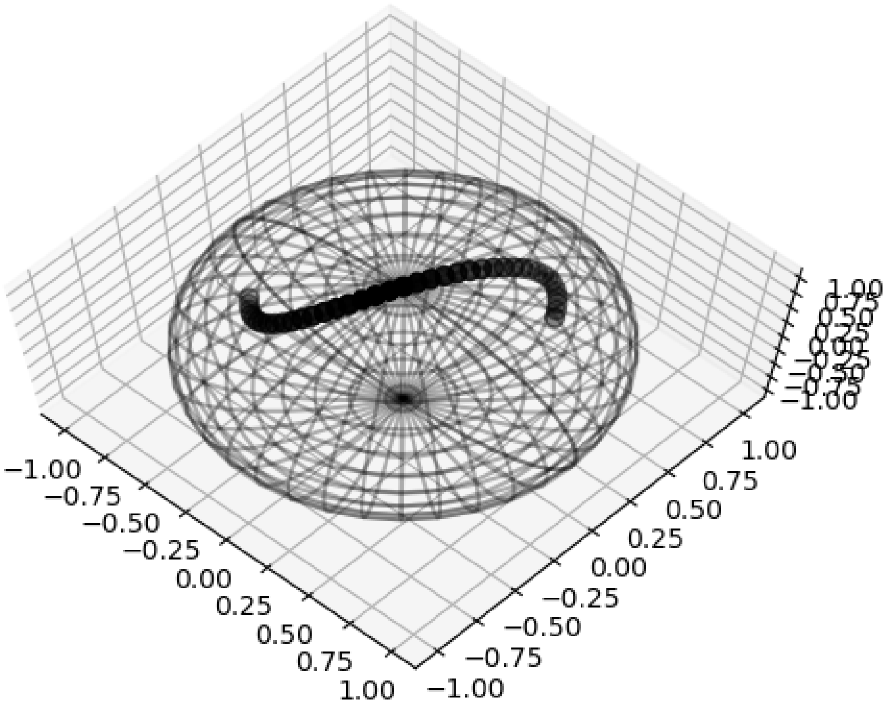
0.8 Algorithm

Applications

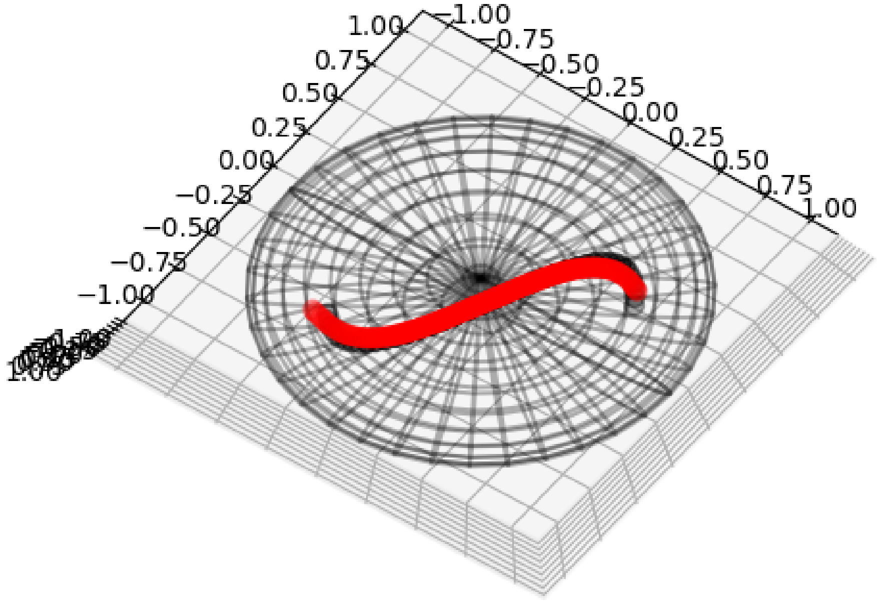
Talk about how data is generated

0.8.1 Toy Data

First we generate some toy data



With the principal flow in red.



We can see that the principal flow accurately reconstructs the toy data, the s curve on the sphere.

0.8.2 Noisy Toy Data

0.8.3 Real World Data

Future Directions

What open questions can we investigate further?

References

Bibliography

[1] Ingwer Borg and Patrick J.F. Groenen (2005) *Modern Multidimensional Scaling*, Springer.

[2] P. Thomas Fletcher, Conglin Lu, Stephen M. Pizer and Sarang Joshi *Principal Geodesic Analysis for the Study of Nonlinear Statistics of Shape*

[3]