# *Estimating the square root of probability density function on Riemannian manifold*

Article

Accepted Version

# *Estimating the Square Root of Probability Density Function on Riemannian Manifold*

Article

# Estimating the Square Root of Probability Density Function on Riemannian Manifold

Xia Hong and Junbin Gao

**Abstract** We propose that the square root of a probability density function can be represented as a linear combination of Gaussian kernels. It is shown that, if the Gaussian kernel centres and kernel width are known, then the maximum likelihood parameter estimator can be formulated as a Riemannian optimization problem on sphere manifold. The first order Riemannian geometry of the sphere manifold and vector transport are initially explored, then the well-known Riemannian conjugate gradient algorithm is used to estimate the model parameters. For completeness the k-means clustering algorithm and a grid search are employed to determine the centers and kernel width respectively. Simulated examples are employed to demonstrate that the proposed approach is effective in constructing the estimate of the square root of probability density function.

## 1 Introduction

The importance of probability density function (PDF) is evidenced by intensive theoretic researches and many data analysis and pattern recognition applications [McLachlan and Peel, 2000, Silverman, 1986, Duda and Hart, 1973, Chen et al., 2010, Rutkowski, 2004, Yin and Allinson, 2001, Dempster et al., 1977, Parzen, 1962]. The Parzen window estimator (PW) and the finite mixture models, especially a mixture of Gaussians, are two widely researched and popular estimators. The mixture of Gaussians model represents an underlying PDF as a linear combination of Gaussian kernels, and the maximum likelihood (ML) estimator of the mixture models

Xia Hong
Department of Computer Science, School of Mathematical and Physical Sciences, University of Reading, UK, e-mail: x.hong@reading.ac.uk

Junbin Gao
Discipline of Business Analytics, The University of Sydney Business School, The University of Sydney, NSW 2006, Australia, e-mail: junbin.gao@sydney.edu.au

parameters can be obtained using the expectation- maximization (EM) algorithm. Alternatively, the Parzen window (PW) estimator [Parzen, 1962] can be regarded as a special case of the finite mixture model [McLachlan and Peel, 2000], in which the number of mixtures is equal to that of the training data samples and all the mixing weights are equal, providing as a simple practical PDF estimator. However if the number of training data samples is very large, then the point density estimate using the PW estimator for a future data sample can be computationally expensive. The associated ML optimisation in a general finite mixture Gaussian is generally a highly nonlinear optimisation process requiring extensive computation, while the EM algorithm for Gaussian mixture models enjoys an explicit iterative form [Bilmes, 1998]. There are also considerable interests into research on sparse PDF estimation which can be summarized into two categories. The first category is based on constrained optimization [Weston et al., 1999, Vapnik and Mukherjee, 2000, Girolami and He, 2003, Hong et al., 2015]. The second category of sparse kernel density estimators construct the PDF estimator in a forward regression manner [Choudhury, 2002,Chen et al., 2004b, Chen et al., 2004a, Chen et al., 2008, Hong et al., 2013].

The Riemannian optimisation algorithms have been recently researched on many types of matrix manifolds such as the Stiefel manifold, Grassmann manifold and the manifold of positive definite matrices, see Section 3.4 of [Absil et al., 2008]. Since Riemannian optimisation is directly based on the curved manifolds, one can eliminate those constraints such as orthogonality to obtain an unconstrained optimisation problem that, by construction, will only use feasible points. This allows one to incorporate Riemannian geometry in the resulting optimisation problems, thus producing far more accurate numerical results. The Riemannian optimisation have been successfully applied in machine learning, computer vision and data mining tasks, including fixed low rank optimisation [Mishra et al., 2013], Riemannian dictionary learning [Harandi et al., 2014], and computer vision [Lui, 2012]. Since the constraint on the mixing coefficients of the finite mixture model is the multinomial manifold, recently we have introduced Riemannian trust-region (RTR) algorithm for sparse finite mixture model based on minimal integrated square error criterion [Hong et al., 2015].

Note that all the aforementioned PDF estimation algorithms are aimed at directly estimating the probability density function. However in this work we investigate the less studied problem of estimating the square root of the probability density function (PDF) estimation [Pinheiro and Vidakovic, 1998], i.e., the square root of a PDF, rather than the PDF itself is a mixture of Gaussians. Clearly the resultant estimator can be used similarly in many applications as can a PDF estimator. However the estimation of the square root of the probability density function (PDF) poses as a different problem, and new estimation algorithm is required.

In this paper, we introduce/extend a fast maximum likelihood estimator based on a linear combination of Gaussian kernels which represents the square root of probability density function [Hong and Gao, 2016]. Initially we apply the k-means clustering algorithm and a grid search to determine the centers and kernel width. It is shown that the underlying parameter estimation problem can be formulated as a Riemannian optimization on the sphere manifold. The first order Riemannian geom-

etry of the sphere manifold and vector transport are explored, and the well-known Riemannian conjugate gradient algorithm is used to estimate the model parameters. Numerical examples are employed to demonstrate that the proposed approach is effective in constructing the estimate of the square root of probability density function.

## 2 Preliminary on Sphere Manifold

This section briefly introduces the concept of sphere manifold and the necessary ingredients used in the retraction based framework of Riemannian optimization. As a reference, the main notations on Riemannian geometry on sphere manifold in this section is summarized in Table 1. We refer the readers to [Absil et al., 2008] for the general concepts of manifolds.

**Table 1** Notations for Sphere Manifold

| | |
|---|---|
| $\{\mathbb{S}^{M-1}, g\}$ | Sphere manifold for parameter matrix $\theta$ and the inner product of the manifold |
| $T_\theta \mathbb{S}^{M-1}$ | Tangent space of the sphere manifold |
| $\mathbf{u}_\theta, \mathbf{v}_\theta$ | Tangent vectors at $\theta$ |
| $\mathrm{Proj}_\theta(\mathbf{z})$ | Orthogonal projector from a vector in ambient space onto the tangent space at $\theta$ |
| $\mathrm{grad}F(\theta)$ | Riemannian gradient of $F(\theta)$ on the manifold $\mathbb{S}^{M-1}$ |
| $\mathrm{Grad}F(\theta)$ | Classical gradient of $F(\theta)$ as seen in Euclidean space |
| $\mathrm{Exp}_\theta$ | Retraction mapping |
| $\mathscr{T}_{\theta_{k+1} \leftarrow \theta_k}(\mathbf{u}_{\theta_k})$ | Vector transport |

The sphere manifold is the set of unit Frobenius norm vectors of size $M$, denoted as

$$\mathbb{S}^{M-1} = \left\{ \theta \in \mathbb{R}^M : \|\theta\|_2 = 1 \right\}. \tag{1}$$

It is endowed with a Riemannian manifold structure by considering it as a Riemannian submanifold of the embedding Euclidean space $\mathbb{R}^M$ endowed with the usual inner product

$$g(\mathbf{u}_\theta, \mathbf{v}_\theta) = \mathbf{u}_\theta^{\mathrm{T}} \mathbf{v}_\theta, \tag{2}$$

where $\mathbf{u}_\theta, \mathbf{v}_\theta \in T_\theta \mathbb{S}^{M-1} \subset \mathbb{R}^M$ are tangent vectors to $\mathbb{S}^{M-1}$ at $\theta$. The inner product on $\mathbb{S}^{M-1}$ determines the geometry such as distance, angle, curvature on $\mathbb{S}^{M-1}$. Note that the tangent space $T_\theta \mathbb{S}^{M-1}$ at element $\theta$ can be described by

$$T_\theta \mathbb{S}^{M-1} = \left\{ \mathbf{u}_\theta : \mathbf{u}_\theta^{\mathrm{T}} \theta = 0 \right\}. \tag{3}$$

*Riemannian gradient*: Let the Riemannian gradient of a scalar function $F(\theta)$ on $\mathbb{S}^{M-1}$ be denoted by $\text{grad}F(\theta)$, and its classical gradient as seen in the Euclidean space as $\text{Grad}F(\theta)$. Then we have

$$\text{grad}F(\theta) = \text{Proj}_\theta\big(\text{Grad}F(\theta)\big), \tag{4}$$

where $\text{Proj}_\theta(\mathbf{z})$ is the orthogonal projection onto the tangent space, which can be computed as

$$\text{Proj}_{\mathbf{X}}(\mathbf{z}) = \mathbf{z} - (\theta^{\mathrm{T}}\mathbf{z})\theta \tag{5}$$

in which $\mathbf{z}$ represents a vector in the ambient space.

*Retraction mapping*: An important concept in the recent retraction-based framework of Riemannian optimization is the retraction mapping, see Section 4.1 of [Absil et al., 2008]. The exponential map $\text{Exp}_{\mathbf{X}}$, defined by

$$\text{Exp}_\theta\big(\lambda\mathbf{u}_\theta\big) = \cos\big(\|\lambda\mathbf{u}_\theta\|_2\big)\theta + \frac{\sin(\|\lambda\mathbf{u}_\theta\|_2)}{\|\mathbf{u}_\theta\|_2}\mathbf{u}_\theta, \tag{6}$$

is the canonical choice for the retraction mapping, where the scalar $\lambda$ is a chosen step size. The retraction mapping is used to locate the next iterate on the manifold along a specified tangent vector, such as a search direction in line search in the Newton's algorithm or the suboptimal tangent direction in the trust-region algorithm, see Chapter 7 of [Absil et al., 2008]. For example, the line search algorithm is simply given by

$$\theta_{k+1} = \text{Exp}_{\theta_k}\big(\lambda_k\mathbf{u}_{\theta_k}\big). \tag{7}$$

where the search direction $\mathbf{u}_{\theta_k} \in T_{\theta_k}\mathbb{S}^{M-1}$ and $\lambda_k$ is a chosen step size at iteration step $k$.

*Vector Transport*: In Riemannian optimization algorithms, the second derivatives can be approximated by comparing the first-order information (tangent vectors) at distinct points on the manifold. The notion of vector transport $\mathscr{T}_{\theta_{k+1}\leftarrow\theta_k}(\mathbf{u}_{\theta_k})$ on a manifold, roughly speaking, specifies how to transport a tangent vector $\mathbf{u}_{\theta_k}$ from a point $\theta_k$ to another point $\theta_{k+1}$ on the manifold. The vector transport for the sphere manifold is calculated as

$$\mathscr{T}_{\theta_{k+1}\leftarrow\theta_k}(\mathbf{u}_{\theta_k}) = \text{Proj}_{\theta_{k+1}}(\mathbf{u}_{\theta_k}) \tag{8}$$

## 3 Proposed estimator for the square root of probability density function

Given the finite data set $D_N = \{\mathbf{x}_j\}_{j=1}^N$ consisting of $N$ data samples, where the data $\mathbf{x}_j \in \mathbb{R}^m$ follows an unknown PDF $p(\mathbf{x})$, the Gaussian mixture model [McLachlan and Peel, 2000] is in the form of

$$p_{GMM}(\mathbf{x}) = \sum_{i=1}^{P} \frac{g_i}{(2\pi)^{m/2}|\Sigma_i|^{m/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu_i)^{\mathrm{T}}\Sigma_i^{-1}(\mathbf{x}-\mu_i)\right) \tag{9}$$

where $\mu_i$ and $\Sigma_i$ are called the mean vector, and covariance matrix (positive definite) of the $i$th mixture. The mixing coefficients satisfies $0 \leq g_i \leq 1$ for all $i$, and

$$\sum_{i=1}^{P} g_i = 1 \tag{10}$$

In this work we investigate the less studied problem of estimating the square root of the probability density function (PDF) estimation [Pinheiro and Vidakovic, 1998]. The reason that the root square of the probability density function is estimated is that the problem can be formulated as a Riemannian optimisation one for computational advantage, as well as that the resultant estimator can be used as an alternative to a probability density function estimator. Specifically, the problem under study is to find a sparse approximation of the square root of $\psi(\mathbf{x}) = \sqrt{p(\mathbf{x})} > 0$ using $M$ component Gaussian kernels, given by

$$\psi(\mathbf{x}) = \sum_{i=1}^{M} \omega_i K_\sigma(\mathbf{x}, \mathbf{c}_i) = \omega^{\mathrm{T}}\mathbf{k}(\mathbf{x}) \tag{11}$$

subject to

$$\int \psi^2(\mathbf{x}) d\mathbf{x} = 1 \tag{12}$$

where

$$K_\sigma(\mathbf{x}, \mathbf{c}_i) = \frac{1}{\left(2\pi\sigma^2\right)^{m/2}} \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{2\sigma^2}\right), \tag{13}$$

in which $\mathbf{c}_i = \left[c_{i,1}\ c_{i,2}\cdots c_{i,m}\right]^{\mathrm{T}}$ is the center vector of the $i$th kernels and $\sigma > 0$ is the width parameter, and $\omega_i$ is the $i$th kernel weight. $\omega = [\omega_1 \cdots \omega_M]^{\mathrm{T}}$, $\mathbf{k}(\mathbf{x}) = [K_\sigma(\mathbf{x}, \mathbf{c}_1) \cdots K_\sigma(\mathbf{x}, \mathbf{c}_M)]^{\mathrm{T}}$.

Applying (11) to the constraint (12), we have

$$\begin{aligned} \int \psi^2(\mathbf{x}) d\mathbf{x} &= \sum_{i=1}^{M}\sum_{j=1}^{M} \omega_i\omega_j \int K_\sigma(\mathbf{x}, \mathbf{c}_i) K_\sigma(\mathbf{x}, \mathbf{c}_j) d\mathbf{x} \\ &= \omega^{\mathrm{T}} Q \omega = 1 \end{aligned} \tag{14}$$

where $Q = \{q_{i,j}\}$, with $q_{i,j} = K_{\sqrt{2}\sigma}(\mathbf{c}_i, \mathbf{c}_j)$, as shown in Appendix A.

Clustering algorithms can be used to find a set of centers which accurately reflects the distribution of the data points. From $N$ data points $\mathbf{x}_j$, $j = 1, \cdots, N$, the k-means algorithm [Haykin, S., 2009] seeks to partition the data points in $M$ disjoint subset $S_i$, each containing $N_i$ data points, so as to minimize the sum-of-squares clustering function given by

$$J = \sum_{i=1}^{M} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \mathbf{c}_i\|^2 \tag{15}$$

where $\in$ denotes belongs to. $J$ is minimized when

$$\mathbf{c}_i = \frac{1}{N_i} \sum_{\mathbf{x}_j \in S_i} \mathbf{x}_j \tag{16}$$

The k-means algorithm is detailed in Algorithm 1.

**Algorithm 1** The k-means clustering algorithm

**Require:** $\mathbf{x}_j$, $j = 1, \cdots, N$, and a preset number of centers $M$.
**Ensure:** $\mathbf{c}_i$, $j = 1, \cdots, M$, that yields the minimum of $J = \sum_{i=1}^{M} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \mathbf{c}_i\|^2$

1: Randomly select $M$ data points from $D_N$ as initial centers $\mathbf{c}_i^{old}$.
2: Randomly draw a data point $\mathbf{x}_j$ from $D_N$.
3: From all $\mathbf{c}_i^{old}$, $i = 1, \cdots, M$, find the nearest center to $\mathbf{x}_j$, denoted as $\mathbf{c}_k^{old}$.
4: Update $\mathbf{c}_k^{new} = \mathbf{c}_k^{old} + \varepsilon(\mathbf{x}_j - \mathbf{c}_k^{old})$, where $\varepsilon > 0$ is the predetermined learning rate.
5: Set $\mathbf{c}_i^{new}$ as $\mathbf{c}_i^{old}$.
6: Goto Step 2 until a sufficient large number of iterations has reached (for convergence).
7: Return $\mathbf{c}_i^{new}$ as $\mathbf{c}_i$, $i = 1, \cdots, M$.

We now propose a new maximum likelihood estimation for parameters $\omega_i$ which is based on that known $M$ kernel centers and a preset kernel width is given.

Initially denote the eigenvalue decomposition $\mathbf{Q}$ as $\mathbf{Q} = \mathbf{U}\Sigma\mathbf{U}^{\mathrm{T}}$, where $\mathbf{U}$ is an orthogonal matrix consisting of the eigenvectors and $\Sigma$ is a diagonal matrix of which the entries are eigenvalues $\mathbf{Q}$.

Let $\theta = \sqrt{\Sigma}\mathbf{U}^{\mathrm{T}}\omega$, $\psi(\mathbf{x})$ can be written as

$$\psi(\mathbf{x}) = \theta^{\mathrm{T}}\bar{\mathbf{k}}(\mathbf{x}) \tag{17}$$

where $\bar{\mathbf{k}}(\mathbf{x}) = \sqrt{\Sigma}^{-1}\mathbf{U}^{\mathrm{T}}\mathbf{k}(\mathbf{x})$.

In order to satisfy $\psi(\mathbf{x}) > 0$, we initialize all $\omega_i$ as the same positive number $\xi$, and $\omega_{Ini} = \xi\mathbf{1}_M$, $\mathbf{1}_M$ is a length $M$ vector with all elements as ones. The corresponding $\theta_0$ can be calculated as

$$\theta_0 = \sqrt{\Sigma}\mathbf{U}^{\mathrm{T}}\mathbf{1}_M \tag{18}$$

followed by a normalization step $\theta_0 = \theta_0/\|\theta_0\|$ so that $\theta_0$ is on the sphere manifold. We are now ready to formulate the maximum likelihood estimator as the following Riemannian optimization problem, given as

$$\theta^{\mathrm{opt}} = \min_{\theta \in \mathbb{S}^{M-1}} \left\{ F(\theta) = -\sum_{j=1}^{N} \log(\theta^{\mathrm{T}}\bar{\mathbf{k}}(\mathbf{x}_j)) \right\} \tag{19}$$

followed by setting $\theta^{\mathrm{opt}} = \sqrt{\Sigma}\mathbf{U}^{\mathrm{T}}\omega^{\mathrm{opt}}$. For our objective function $F(\theta)$, it is easy to check that Euclidean gradient Grad$F(\theta)$ can be calculated respectively as

$$\text{Grad}F(\theta) = -\sum_{j=1}^{N} \frac{\bar{\mathbf{k}}(\mathbf{x}_j)}{\theta^{\mathrm{T}}\bar{\mathbf{k}}(\mathbf{x}_j)} \tag{20}$$

Based on Grad$F(\theta)$, Riemannian gradient of the objective function $F(\theta)$ on the sphere manifold can be calculated according to (4), (5). We opt to Riemannian conjugate gradient algorithm to solve (19), which generalizes the classical conjugate gradient algorithm [Hager and Zhang, 2006] to optimization problems over Riemannian manifolds [Boumal et al., 2014]. Since the logarithm function acts as a natural barrier at zero and the Riemannian conjugate gradient algorithm is a local minimization algorithm, the constraint $\psi(\mathbf{x}) > 0$ can be met.

With all the ingredients available, we form the algorithm for solving (19) in Algorithm 1, which is well implemented in the Manifold Optimization Toolbox Manopt `http://www.manopt.org`, see [Boumal et al., 2014]. We used the default parameter settings in Manopt, so that in Step 4 of Algorithm 1 $\alpha_k$ is based on line search backtracking procedure as described in Chapter 4, p63 of Section 4 of [Absil et al., 2008]. In Step 5 of Algorithm 1 $\omega_{k+1}$ is based on the default option "Hestenes-Stiefel's modified rule". Specifically, we have

$$\beta_{k+1} = \max\left\{0, \frac{< \text{grad}F(\theta_{k+1}), \gamma_{k+1} >}{< \mathscr{T}_{\theta_{k+1}\leftarrow\theta_k}(\eta_k), \gamma_{k+1} >}\right\} \tag{21}$$

where $< \cdot, \cdot >$ denotes inner product, and

$$\gamma_{k+1} = \text{grad}F(\theta_{k+1}) - \mathscr{T}_{\theta_{k+1}\leftarrow\theta_k}(\text{grad}F(\theta_k)) \tag{22}$$

**Algorithm 2** Riemannian conjugate gradient Algorithm for solving (19)

**Require:** $\bar{\mathbf{k}}(\mathbf{x}_j), j = 1, \cdots, N$. Initial point $\theta_0$ which is on the sphere manifold $\mathbb{S}^{M-1}$, and default threshold $\varpi$, e.g., $\varpi = 10^{-6}$ or $\varpi = 10^{-5}$;

**Ensure:** $\theta$ that yields the minimum $F(\theta)$.

1: Set $\eta_0 = -\text{grad}F(\theta_0)$ and $k = 0$;
2: **while** $\|\text{grad}F(\theta_k)\|_2 < \varpi$ **do**
3: $k = k + 1$;
4: Compute a step size $\alpha_k$ and set

$$\theta_k = \text{Exp}_{\theta_{k-1}}(\alpha_k \eta_{k-1}); \tag{23}$$

5: Compute $\beta_k$ and set

$$\eta_k = -\text{grad}F(\theta_k) + \beta_k \mathscr{T}_{\theta_k\leftarrow\theta_{k-1}}(\eta_{k-1}); \tag{24}$$

6: **end while**
7: Return $\theta = \theta_k$, $F(\theta)$, and the associated $\mathbf{U}$, $\Sigma$.

In summary our proposed algorithm consists of two consecutive steps; (i) at the first stage, we determine $M$ kernel centres using the k-means clustering algorithm, as presented in Algorithm 1; (ii) for a grid search of kernel width, estimate kernel parameters using Riemannian conjugate gradient algorithm $\omega$ based on the maxi-

**Algorithm 3** Proposed estimator for the square root of probability density function

**Require:** $\mathbf{x}_j, j = 1, \cdots, N$. Preset number of kernels $M$;
**Ensure:** $\psi(\mathbf{x})$ that maximizes the likelihood.
1: Use the k-means clustering algorithm (Algorithm 1) to find $\mathbf{c}_i, i = 1, \cdots, M$;
2: **for** $i = 1, 2, ..., (Iter + 1)$ **do**
3: $\sigma = \sigma_{min} + \frac{i-1}{n}(\sigma_{max} - \sigma_{min})$. where $(Iter + 1), \sigma_{min}$ and $\sigma_{max}$ are preset, and they denote the minimum, maximum value and the total number of kernel width on a grid search.
4: Form $\mathbf{Q}$ and its eigen-decomposition $\mathbf{Q} = \mathbf{U}\Sigma\mathbf{U}^{\mathrm{T}}$
5: Obtain $\tilde{\mathbf{k}}(\mathbf{x}_j) = \sqrt{\Sigma}^{-1}\mathbf{U}^{\mathrm{T}}\mathbf{k}(\mathbf{x}_j), j = 1, \cdots, N$
6: Find $\theta_0$ according to (18) and normalize it.
7: Apply the Riemannian conjugate gradient Algorithm (Algorithm 2) to return $F(\theta)$.
8: **end for**
9: Find $\sigma^{\mathrm{opt}}$ that minimizes $F(\theta)$ over the grid search. Return the associated $\theta$, $\mathbf{U}$, $\Sigma$.
10: Return $\omega^{\mathrm{opt}} = \mathbf{U}\sqrt{\Sigma}^{-1}\theta^{\mathrm{opt}}$.

mum likelihood criterion. For completeness, the proposed estimator is detailed in Algorithm 3.

The computational complexity consists of the k-means clustering which is in the order $Iter_{kmeans} * O(N)$, where $Iter_{kmeans}$ is the number of iterations of k-means clustering algorithm, and that of the Riemannian conjugate algorithm, scaled by the number of grid search $Iter$. The main cost in Riemannian conjugate algorithm is function and derivative evaluation of (19)& (20) which are in the order of $O(MN)$. Hence the total cost of the proposed algorithm is evaluated as $Iter_{kmeans} * O(N) + Iter * O(MN)$.

## 4 Illustrative examples

Example 1: A data set of $N = 600$ points was randomly drawn from a known distribution $p(\mathbf{x})$ and used to construct the estimator of the square root of probability density function $\psi(\mathbf{x})$ using the proposed approach. $p(\mathbf{x})$ is given as a mixture of one Gaussian and one Laplacian, as defined by

$$p(\mathbf{x}) = \frac{2}{3\pi}\exp\left(-2(x_1-1)^2 + 2(x_2-1)^2\right) + \frac{0.35}{6}\exp\left(-0.7|x_1+1| - 0.5|x_2+1|\right) \tag{25}$$

We preset $M = 40$, the k-means algorithm was applied to find $M = 40$ centers $\mathbf{c}_i, i = 1, \cdots, M$. The Riemannian conjugate gradient algorithm was applied to minimize the negative likelihood based on $\mathbf{Q}$ obtained a range of kernel widths of $[0.5, 2]$, with a grid width of 0.1. The result of $\log F(\theta)$ versus the kernel width was shown in Figure 1. The convergence of Riemannian conjugate gradient algorithm on the sphere manifold based on the optimal kernel width was shown shown in Figure 2, showing that the algorithm can converge rapidly. The performance of proposed es-

timator was shown in Figure 3(a),(b)&(c), which plots $\psi^2(\mathbf{x})$, the true density $p(\mathbf{x})$, and the estimation error $e(\mathbf{x}) = \psi(\mathbf{x})^2 - p(\mathbf{x})$ respectively over a $41 \times 41$ meshed data, with a grid size of 0.2, ranging from -4 to 4 for both $x_1$ and $x_2$.

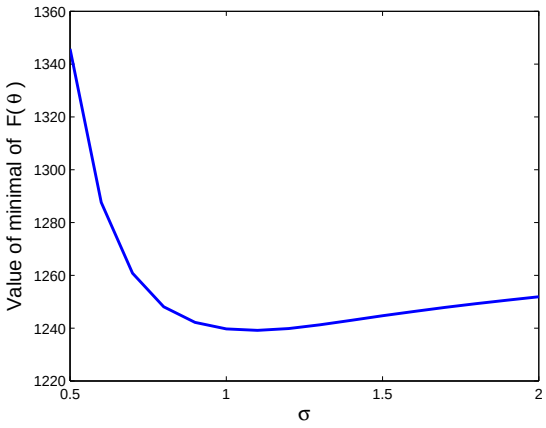



**Fig. 1** $\log(F(\theta))$ versus the kernel width for Example 1.

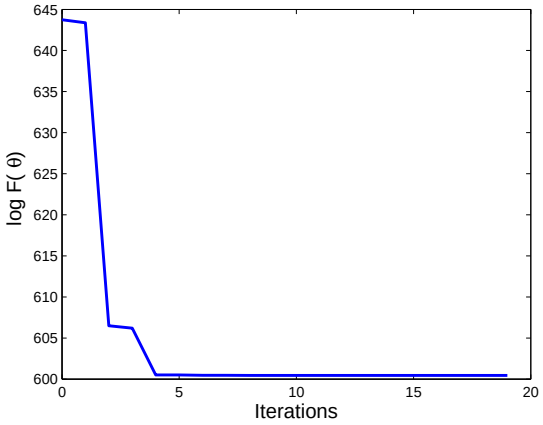
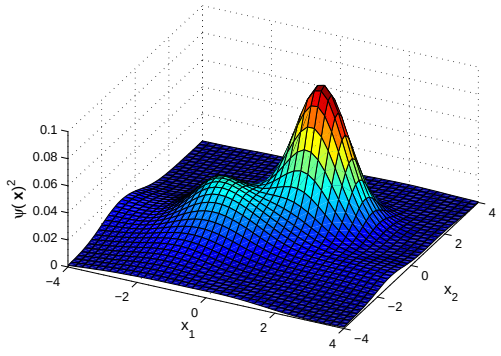
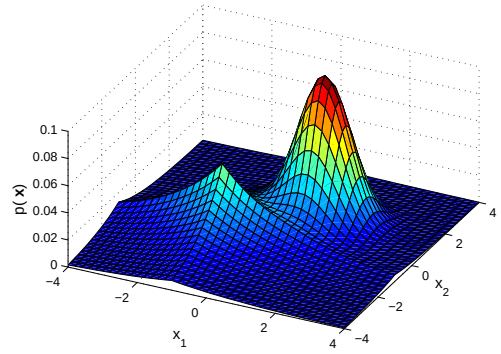



**Fig. 2** Convergence of Riemannian conjugate gradient algorithm for Example 1.

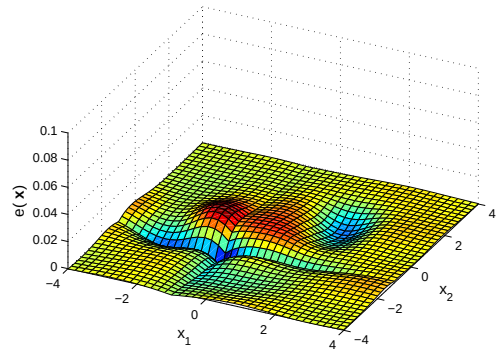
The experiment was repeated for 100 different random runs in order to compare the performance of the proposed algorithm with the Gaussian mixture model (GMM) that is fitted using the EM algorithm [McLachlan and Peel, 2000]. A sepa-




(a)




(b)




(c)

**Fig. 3** The result of the proposed estimator for Example 1; (a) The resultant pdf estimator $\psi(\mathbf{x})^2$ (b) The true pdf $p(\mathbf{x})$; and (c)The estimation error $e(\mathbf{x})$.

rate test data set of $N_{test} = 1681$ was generated using a $41 \times 41$ meshed data, with a grid size of 0.2, ranging from -4 to 4 for both $x_1$ and $x_2$. These points were used for evaluation according to

$$L_1 = \begin{cases} \frac{1}{N_{test}} \sum_{k=1}^{N_{test}} |p(\mathbf{x}_k) - \psi^2(\mathbf{x}_k)| & \text{Proposed} \\ \frac{1}{N_{test}} \sum_{k=1}^{N_{test}} |p(\mathbf{x}_k) - p_{\text{GMM}}(\mathbf{x}_k)| & \text{GMM} \end{cases} \tag{26}$$

where $p_{\text{GMM}}$ is the resultant GMM model based on two Gaussian mixtures, with the constraint that the covariance matrix is diagonal. The GMM model fitting is implemented using Matlab command *gmdistribution.fit.m*. The results are as shown Table 2 which demonstrate that the proposed algorithm outperforms the GMM fitted using EM algorithm with two mixtures.

**Table 2** Performance of proposed density estimator in comparison with GMM model for Example 1.

| Method | $L_1$ test error (mean $\pm$ STD) |
|---|---|
| Proposed | $(2.2 \pm 0.3) \times 10^{-4}$ |
| GMM | $(3.2 \pm 0.6) \times 10^{-4}$ |

Example 2: A data set of $N = 1500$ points was randomly drawn from a known distribution $p(\mathbf{x})$ given as a mixture of one Gaussian and one Laplacian, as defined by

$$\begin{aligned} p(\mathbf{x}) = & \frac{1}{2.1\pi} \exp\left(-2(x_1-2)^2 - (x_2-2)^2/0.98\right) \\ & + \frac{0.35}{6} \exp\left(-0.7|x_1+1| - 0.5|x_2+1|\right) \end{aligned} \tag{27}$$

and the proposed approach is experimented to estimate $\psi(\mathbf{x})$, the square root of $p(\mathbf{x})$.

The number of centers are empirically set as $M = 40$, the k-means algorithm was implemented according to Algorithm 1, producing $M$ centers $\mathbf{c}_i, i = 1, \cdots, M$. Based on the centers, the kernel matrices $\mathbf{Q}$ are generated using any kernel width on the grid point on $[0.2, 2]$, with a grid width of 0.1. The Riemannian conjugate gradient algorithm was used to minimize the negative likelihood for the range of kernel widths. The result of $\log F(\theta)$ versus the kernel width was shown in Figure 4. The convergence of Riemannian conjugate gradient algorithm on the sphere manifold based on the optimal kernel width was shown shown in Figure 5, showing that the algorithm can converge rapidly. The performance of proposed estimator was shown in Figure 6(a),(b)&(c), which plots $\psi^2(\mathbf{x})$, the true density $p(\mathbf{x})$, and the estimation error $e(\mathbf{x}) = \psi(\mathbf{x})^2 - p(\mathbf{x})$ respectively over a $41 \times 41$ meshed data, with a grid size of 0.2, ranging from -4 to 4 for both $x_1$ and $x_2$.

The experiment was repeated for 100 different random runs in order to compare the performance of the proposed algorithm with the Gaussian mixture model

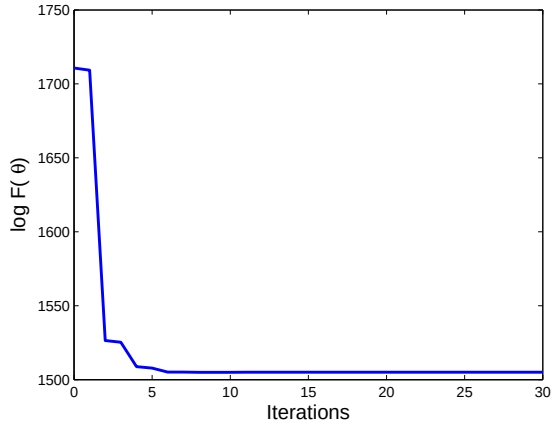



**Fig. 4** $\log(F(\theta))$ versus the kernel width for Example 2.

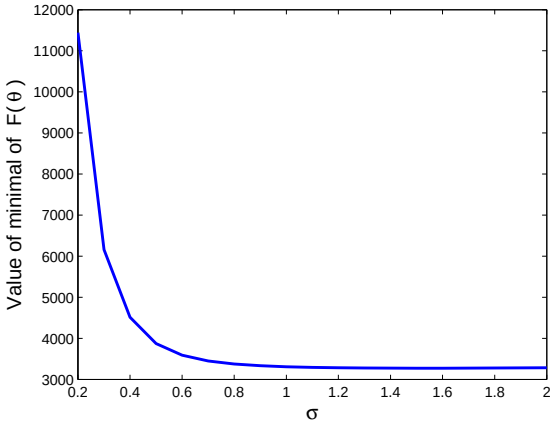



**Fig. 5** Convergence of Riemannian conjugate gradient algorithm for Example 2.

(GMM) that is fitted using the EM algorithm [McLachlan and Peel, 2000]. A separate test data set of $N_{test} = 1681$ was generated using a $41 \times 41$ meshed data, with a grid size of 0.2, ranging from -4 to 4 for both $x_1$ and $x_2$. These points were used for evaluation according to

$$L_1 = \begin{cases} \frac{1}{N_{test}} \sum_{k=1}^{N_{test}} |p(\mathbf{x}_k) - \psi^2(\mathbf{x}_k)| & \text{Proposed} \\ \frac{1}{N_{test}} \sum_{k=1}^{N_{test}} |p(\mathbf{x}_k) - p_{\text{GMM}}(\mathbf{x}_k)| & \text{GMM} \end{cases} \tag{28}$$

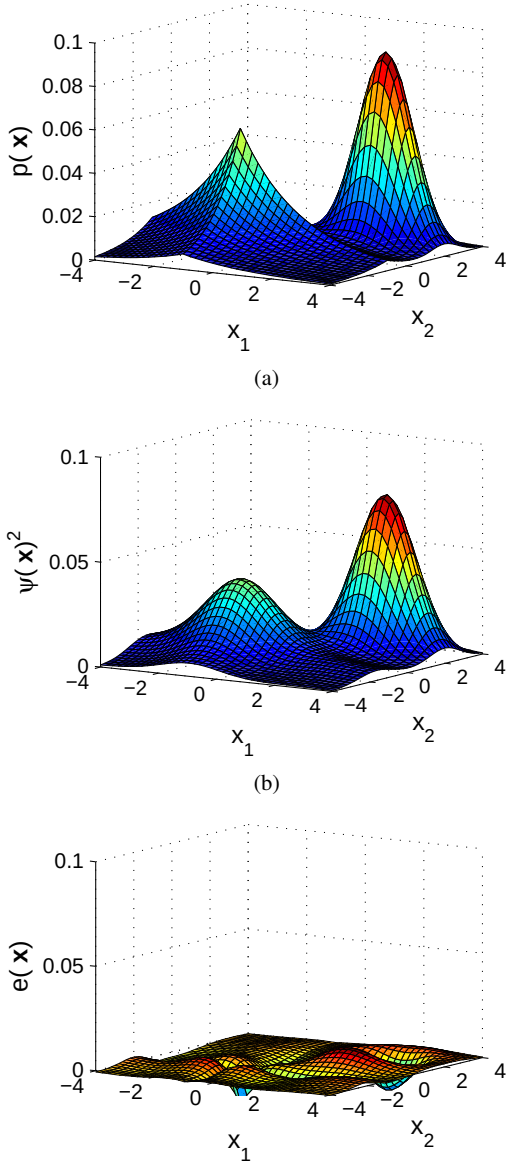



**Fig. 6** The result of the proposed estimator for Example 2; (a) The resultant pdf estimator $\psi(\mathbf{x})^2$ (b) The true pdf $p(\mathbf{x})$; and (c)The estimation error $e(\mathbf{x})$.

where $\hat{p}_{\text{GMM}}$ is the resultant GMM model, with the constraint that the covariance matrix is diagonal. The GMM model fitting is implemented using Matlab command $gmdistribution.fit.m$ with two mixtures. The results are as shown Table 3, showing that the proposed method is better than Gaussian mixture models with two mixtures.

**Table 3** Performance of proposed density estimator in comparison with GMM model for Example 2.

| Method | $L_1$ test error (mean $\pm$ STD) |
|---|---|
| Proposed | $(1.7 \pm 0.2) \times 10^{-3}$ |
| GMM | $(2.9 \pm 0.1) \times 10^{-3}$ |

## 5 Conclusions

In this paper a new method has been introduced to estimate the square root of probability density function from observational data using maximum likelihood. The proposed model is in the form of a linear combination of Gaussian kernels. Incorporating the k-means clustering algorithm to determine the kernel center as well as a grid search to determine the kernel width, the model parameters are then estimated using the Riemannian optimization on the sphere. The first order Riemannian geometry of the sphere manifold and vector transport are explored. Two illustrative examples are employed to demonstrate that models obtained by the proposed approach are comparable with the GMM model fitted using EM algorithm.

In this research we have included the well-known k-means clustering algorithm which is based on a predetermined number $M$ centers which may not optimal. The choice of $M$ is generally dependent on applications, and for k-means clustering algorithm, $M$ should be set sufficiently large to perform well. Future research will be focused on model structure optimization so that the model size can be learnt from data.

## Appendix A

Let $\mathbf{x} = [x_1, ....x_m]^{\text{T}}$, we have

$$
\begin{aligned}
q_{i,j} &= \frac{1}{(2\pi\sigma^2)^m}\int\ldots\int \exp\Big(-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{2\sigma^2}-\frac{\|\mathbf{x}-\mathbf{c}_j\|^2}{2\sigma^2}\Big)\\
&\quad dx_1\ldots dx_m\\
&= \frac{1}{(2\pi\sigma^2)^m}\prod_{l=1}^{m}\int \exp\Big(-\frac{(x_l-c_{i,l})^2}{2\sigma^2}-\frac{(x_l-c_{j,l})^2}{2\sigma^2}\Big)dx_l
\end{aligned}
\tag{29}
$$

in which

$$
\begin{aligned}
&\int \exp\Big(-\frac{(x_l-c_{i,l})^2}{2\sigma^2}-\frac{(x_l-c_{j,l})^2}{2\sigma^2}\Big)dx_l\\
&= \int \exp\Big(-\frac{x_l^2-(c_{i,l}+c_{j,l})x_l+(c_{i,l}^2+c_{j,l}^2)/2}{\sigma^2}\Big)dx_l\\
&= \exp\Big(-\frac{\frac{c_{j,l}^2+c_{i,l}^2}{2}-\big(\frac{c_{i,l}+c_{j,l}}{2}\big)^2}{\sigma^2}\Big)\\
&\quad\times\int \exp\Big(-\frac{\big[x_l-(c_{i,l}+c_{j,l})\big]^2}{\sigma^2}\Big)dx_l
\end{aligned}
\tag{30}
$$

By making use of $\int \frac{1}{\sqrt{2\pi\sigma^2}}\exp\Big(-\frac{(x_l-c)^2}{2\sigma^2}\Big)dx_l = 1$, i.e. Gaussian density integrates to one, we have

$$
\begin{aligned}
&\int \exp\Big(-\frac{(x_l-c_{i,l})^2}{2\sigma^2}-\frac{(x_l-c_{j,l})^2}{2\sigma^2}\Big)dx_l\\
&= \sqrt{\pi\sigma^2}\exp\Big(-\frac{\frac{c_{j,l}^2+c_{i,l}^2}{2}-\big(\frac{c_{i,l}+c_{j,l}}{2}\big)^2}{\sigma^2}\Big)\\
&= \sqrt{\pi\sigma^2}\exp\Big(-\frac{(c_{j,l}-c_{i,l})^2}{4\sigma^2}\Big)
\end{aligned}
\tag{31}
$$

Hence

$$
q_{i,j} = \frac{1}{(4\pi\sigma^2)^m}\exp\Big(-\frac{\|\mathbf{c}_i-\mathbf{c}_j\|^2}{4\sigma^2}\Big) = K_{\sqrt{2}\sigma}(\mathbf{c}_i,\mathbf{c}_j) \tag{32}
$$

## References


Absil, P.-A., Mahony, R., and Sepulchre, R. (2008). *Optimization Algorithms on Matrix Manifolds*. Princeton University Press.

Bilmes, J. A. (1998). A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical Report ICSI-TR-97-021, University of California, Berkeley.

Boumal, N., Mishra, B., Absil, P.-A., and Sepulchre, R. (2014). Manopt, a matlab toolbox for optimization on manifolds. *J. Machine Learning research*, 15:1455–1459.

Chen, S., Hong, X., and Harris, C. J. (2004a). Sparse kernel density construction using orthogonal forward regression with leave-one-out test score and local regularization. *IEEE Trans. on Systems, Man and Cybernetics, Part B: Cybernetics*, 34(4):1708–1717.

Chen, S., Hong, X., and Harris, C. J. (2008). An orthogonal forward regression technique for sparse kernel density estimation. *Neurocomputing*, 71(4-6):925–943.

Chen, S., Hong, X., and Harris, C. J. (2010). Particle swarm optimization aided orthogonal forward regression for unified data modelling. *IEEE Trans. Evolutionary Computation*, 14:477–499.

Chen, S., Hong, X., Harris, C. J., and Sharkey, P. M. (2004b). Sparse modelling using orthogonal forward regression with PRESS statistic and regularization. *IEEE Trans. on Systems, Man and Cybernetics, Part B: Cybernetics*, 34(2):898–911.

Choudhury, A. (2002). *Fast Machine Learning Algorithms for Large Data*. PhD thesis, School of Engineering Sciences, University of Southampton.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society B*, 39:1–38.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. Wiley: New York, 1973.

Girolami, M. and He, C. (2003). Probability density estimation from optimally condensed data samples. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25(10):1253–1264.

Hager, W. W. and Zhang, H. (2006). A survey of nonlinear conjugate gradient methods. *Pacific J. Optimization*, 2:35–58.

Harandi, M., aqnd C. Shen, R. H., Lovell, B., and Sanderson, C. (2014). Extrinsic methods for coding and dictionary learning on Grassmann manifolds. arXiv:1401.8126.

Haykin, S. (2009). *Neural Networks and Learning Machines*. Pearson Education Inc.

Hong, X., Chen, S., Qatawneh, A., Daqrouq, K., Sheikh, M., and Morfeq, A. (2013). Sparse probability density function estimation using the minimum integrated square error. *Neurocomputing*, 115:122–129.

Hong, X., Gao, J., Chen, S., and Zia, T. (2015). Sparse density estimation on the multinomial manifold. *IEEE Transactions on Neural Networks and Learning Systems*, 26(11):2972–2977.

Hong, X. and Gao, J. B. (2016). A fast algorithm to estimate the square root of probability density function. In *Proceeding of AI-2016, Thirty-sixth SGAI International Conference on Artificial Intelligence*, Cambridge, UK.

Lui, Y. M. (2012). Advances in matrix manifolds for computer vision. *Image and Vision Computing*, 30:380–388.

McLachlan, G. and Peel, D. (2000). *Finite Mixture Models*. Wiley: New York.

Mishra, B., Meyer, G., Bach, F., and Sepulchre, R. (2013). Low-rank optimization with trace norm penalty. *SIAM Journal on Optimization*, 23:2124–2149.

Parzen, E. (1962). On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1066–1076.

Pinheiro, A. and Vidakovic, B. (1998). Estimating the square root of density via compactly supported wavelets. *Computational Statistics and Data Analysis*, 25:399–415.

Rutkowski, L. (2004). Adaptive probabilistic neural networks for pattern classification in time-varying environment. *IEEE Trans. Neural Networks*, 15:811–827.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman and Hall: London.

Vapnik, V. and Mukherjee, S. (2000). Support vector machine for multivariate density estimation. In S. Solla, T. L. and Müller, K. R., editors, *Advances in Neural Infromation Processing Systems*, pages 659–665. MIT Press.

Weston, J., Gammerman, A., Stitson, M., Vapnik, V., Vovk, V., and Watkins, C. (1999). Suppot vector density estimation. In B. Schölkopf, C. B. and Smola, A. J., editors, *Advances in Kernel Methods*, pages 293–306. MIT Press.

Yin, H. and Allinson, N. W. (2001). Self-organizing mixture networks for probability density estimation. *IEEE Trans. Neural Networks*, 12:405–411.