Int J Comput Vis (2009) 81: 317–330
DOI 10.1007/s11263-008-0178-9

Spectral Curvature Clustering (SCC)

Guangliang Chen · Gilad Lerman

Received: 15 November 2007 / Accepted: 12 September 2008 / Published online: 10 December 2008

Abstract This paper presents novel techniques for improving the performance of a multi-way spectral clustering framework (Govindu in Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 1, pp. 1150–1157, 2005; Chen and Lerman, 2007, preprint in the supplementary webpage) for segmenting affine subspaces. Specifically, it suggests an iterative sampling procedure to improve the uniform sampling strategy, an automatic scheme of inferring the tuning parameter from the data, a precise initialization procedure for K-means, as well as a simple strategy for isolating outliers. The resulting algorithm, Spectral Curvature Clustering (SCC), requires only linear storage and takes linear running time in the size of the data. It is supported by theory which both justifies its successful performance and guides our practical choices. We compare it with other existing methods on a few artificial instances of affine subspaces. Application of the algorithm to several real-world problems is also discussed.

Keywords Hybrid linear modeling · Multi-way spectral clustering · Polar curvature · Iterative sampling · Motion segmentation · Face clustering

This work was supported by NSF grant #0612608.
Supplementary webpage: http://www.math.umn.edu/~lerman/scc/.

G. Chen · G. Lerman (✉)
School of Mathematics, University of Minnesota, 127 Vincent Hall, 206 Church Street SE, Minneapolis, MN 55455, USA
e-mail: lerman@umn.edu

G. Chen
e-mail: glchen@math.umn.edu

1 Introduction

We address the problem of *hybrid linear modeling*. Roughly speaking, we assume a data set that can be well approximated by a mixture of affine subspaces, or equivalently, flats, and wish to estimate the parameters of each of the flats as well as the membership of the given data points associated with them. More precise formulations of this problem appear in Ma et al. (2008) and Chen and Lerman (2007).

There are many algorithms that can be applied to this problem. Some of them emphasize modeling of the underlying flats and then use the models to infer the clusters (see e.g., Independent Component Analysis (Hyvärinen and Oja 2000), Subspace Separation (Kanatani 2001, 2002), Generalized Principal Component Analysis (GPCA) (Vidal et al. 2005; Ma et al. 2008)). A few others address the clustering part and then use its output to estimate the parameters of the underlying flats (see e.g., Multi-way Clustering algorithms (Agarwal et al. 2005, 2006; Govindu 2005; Shashua et al. 2006), Tensor Voting (Medioni et al. 2000), k-Manifolds (Souvenir and Pless 2005), Grassmann Clustering (Gruber and Theis 2006), Poisson Mixture Model (Haro et al. 2006)). There are also algorithms that iterate between the two components of data clustering and subspace modeling (see e.g., Mixtures of PPCA (MoPPCA) (Tipping and Bishop 1999), K-Subspaces (Ho et al. 2003)/k-Planes (Bradley and Mangasarian 2000; Tseng 1999)).

In this paper we mainly focus on the special case of hybrid linear modeling where all the flats have the same dimension $d \geq 0$. We emphasize the clustering component, and thus refer to this special case as *d-flats clustering*. We follow Govindu's framework of multi-way spectral clustering (Govindu 2005) and Ng et al.'s framework of spectral clustering (Ng et al. 2002). In our setting, the former framework starts by assigning to any $d + 2$ points in the data

an affinity measure quantifying d-dimensional flatness, thus forming a $(d+2)$-way affinity tensor. Next, it obtains a similarity matrix by decomposing the affinity tensor so that spectral clustering methods can be applied.

However, there are critical issues associated with this framework that need to be thoroughly addressed. First of all, as the size of the data and the intrinsic dimension d increase, it is computationally prohibitive to calculate or store, not to mention process, the affinity tensor. Approximating this tensor by uniformly sampling a small subset of its "fibers" (Govindu 2005) is insufficient for large d and data of moderate size. Better numerical techniques have to be developed while maintaining both reasonable performance and fast speed. Second, the multi-way affinities contain a tuning parameter which crucially affects clustering. It is not clear how to select its optimal value while avoiding an exhaustive search. There are also smaller issues, e.g., how to deal with outliers.

Our algorithm, Spectral Curvature Clustering (SCC), provides specific solutions to the above issues. More specifically, it contributes to the advancement of multi-way spectral clustering in the following aspects.

- It introduces an iterative sampling procedure to significantly improve accuracy over the standard random sampling scheme used in Govindu (2005) (see Sect. 3.1.1).
- It suggests an automatic way of estimating the tuning parameter commonly used in multi-way spectral clustering methods (see Sect. 3.1.2).
- It employs an efficient way of applying K-means in its setting (see Sect. 3.1.3).
- It proposes a simple strategy to isolate outliers while clustering flats (see Sect. 3.4).

Careful analysis of the theoretical performance of the SCC algorithm appears in Chen and Lerman (2007).

The rest of the paper is organized as follows. In Sect. 2 we first introduce our multi-way affinities, and then review a theoretical version of the SCC algorithm (Chen and Lerman 2007). Section 3 discusses various techniques that are used to make the theoretical version practical, and the SCC algorithm is formulated incorporating these techniques. We compare our algorithm with other competing methods using various kinds of artificial data sets as well as several real-world applications in Sect. 4. Section 5 concludes with a brief discussion and possible avenues for future work.

2 Background

2.1 Polar Curvature

Let d and D be integers such that $0 \le d < D$. For any $d+2$ distinct points $\mathbf{z}_1, \ldots, \mathbf{z}_{d+2}$ in $\mathbb{R}^D$, we denote by $V_{d+1}(\mathbf{z}_1, \ldots, \mathbf{z}_{d+2})$ the volume of the $(d+1)$-simplex formed by these points. The polar sine at each vertex $\mathbf{z}_i$ is

$$\begin{aligned}&\operatorname{psin}_{\mathbf{z}_i}(\mathbf{z}_1, \ldots, \mathbf{z}_{d+2})\\ &= \frac{(d+1)! \cdot V_{d+1}(\mathbf{z}_1, \ldots, \mathbf{z}_{d+2})}{\prod_{\substack{1 \le j \le d+2 \\ j \ne i}} \|\mathbf{z}_j - \mathbf{z}_i\|}, \quad 1 \le i \le d+2. \end{aligned} \tag{1}$$

The polar curvature of the $d+2$ points is defined as follows (Chen and Lerman 2007; Lerman and Whitehouse 2008d):

$$\begin{aligned}c_{\mathrm{p}}(\mathbf{z}_1, \ldots, \mathbf{z}_{d+2}) &= \operatorname{diam}(\{\mathbf{z}_1, \ldots, \mathbf{z}_{d+2}\})\\ &\quad \times \sqrt{\sum_{i=1}^{d+2} \left(\operatorname{psin}_{\mathbf{z}_i}(\mathbf{z}_1, \ldots, \mathbf{z}_{d+2})\right)^2}, \end{aligned} \tag{2}$$

where diam(S) denotes the diameter of the set S. We remark that when $d = 0$, the polar curvature coincides with the Euclidean distance.

It is shown in Lerman and Whitehouse (2008d) (following the methods of Lerman and Whitehouse 2008a, 2008b, 2008c) that under certain conditions the least squares error of approximating certain probability measures μ by d-flats is comparable to the average of c_{p}^2 (with respect to μ^{d+2}). This observation is used in the theoretical analysis of the SCC algorithm (Chen and Lerman 2007).

2.2 The Affinity Tensor and its Matrix Representation

We assume a data set $\mathrm{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ in $\mathbb{R}^D$ sampled from a collection of K d-flats (possibly with noise and outliers), where $K > 1$ and N is large. Using the above polar curvature c_{p} and a fixed constant $\sigma > 0$, we construct the following multi-way affinity for any $d+2$ points $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_{d+2}}$ in X:

$$\begin{aligned}&\mathcal{A}(i_1, \ldots, i_{d+2})\\ &= \begin{cases} e^{-c_{\mathrm{p}}^2(\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_{d+2}})/(2\sigma^2)}, & \text{if } i_1, \ldots, i_{d+2} \text{ are distinct;} \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{3}$$

We will explain in Sect. 3.1.2 how to select the optimal value of the tuning parameter σ.

Equation (3) defines a $(d+2)$-way tensor $\mathcal{A}$ of size $N \times N \times \cdots \times N$, but we will only use a matrix representation of $\mathcal{A}$, which we denote by $\mathbf{A}$ and call the affinity matrix. The size of $\mathbf{A}$ is $N \times N^{d+1}$. For each $1 \le i \le N$, the i^{th} row of the matrix $\mathbf{A}$ (i.e., $\mathbf{A}(i, :)$) is expanded from the i^{th} slice of the tensor $\mathcal{A}$ (i.e., $\mathcal{A}(i, :, \ldots, :)$) following some arbitrary but fixed order, e.g., the lexicographic order, of the last $d+1$ indices (see e.g., Bader and Kolda 2004, Fig. 2). This ordering is not important to us, since what we really need is the product $\mathbf{A}\mathbf{A}'$ (see Algorithm 1 below), which is independent of such ordering.

2.3 The SCC Algorithm in Theory

The Theoretical Spectral Curvature Clustering (TSCC) algorithm (Chen and Lerman 2007) is presented below (Algorithm 1) for solving the d-flats clustering problem.

Algorithm 1 Theoretical Spectral Curvature Clustering (TSCC)

Input: $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D$: data, d: dimension, K: number of d-flats, σ: tuning parameter.

Output: K disjoint clusters $C_1, \ldots, C_K$.

Steps:

1: Construct the affinity tensor $\mathcal{A}$ using (3). Unfold $\mathcal{A}$ to obtain an affinity matrix $\mathbf{A}$ and compute $\mathbf{W} = \mathbf{A} \cdot \mathbf{A}'$.

2: Calculate the degrees $\mathbf{D} = \text{diag}\{\mathbf{W} \cdot \mathbf{1}\}$, where $\mathbf{1}$ is the vector of all 1's, and normalize $\mathbf{W}$ using $\mathbf{D}$ to get $\mathbf{Z} = \mathbf{D}^{-1/2} \cdot \mathbf{W} \cdot \mathbf{D}^{-1/2}$.

3: Find the leading K eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_K$ of $\mathbf{Z}$ and define $\mathbf{U} = [\mathbf{u}_1 \mathbf{u}_2 \ldots \mathbf{u}_K] \in \mathbb{R}^{N \times K}$.

4: Apply K-means to group the rows of $\mathbf{U}$ (possibly normalized to have unit length) into K subsets, and correspondingly divide X into K disjoint clusters $C_1, \ldots, C_K$.

We justify this algorithm and give conditions under which it is expected to work well in Chen and Lerman (2007). The first step of the justification is to establish the precise segmentation of the TSCC algorithm in an *ideal* case, where the affinity tensor attains 1 at the entries that correspond to $d+2$ distinct points from a single underlying cluster and is otherwise 0 (this tensor is called *the perfect tensor*). It is then extended to more general cases using perturbation analysis and concentration inequalities. We remark that we will use this ideal case to guide various techniques when developing a practical algorithm in Sect. 3.

Remark 2.1 The question of whether or not to normalize the rows of $\mathbf{U}$ (to have unit length) is an interesting one. We further discuss it in Chen and Lerman (2007). In this paper we choose not to normalize the rows of $\mathbf{U}$. However, the strategies presented in Sect. 3 apply to either case.

2.4 Assessment of Algorithm

We evaluate the performance of the TSCC algorithm using two types of errors: $e_{\text{OLS}}, e_{\%}$. For any K detected clusters $C_1, \ldots, C_K$, the averaged Orthogonal Least Squares (OLS) error is defined as follows:

$$e_{\text{OLS}} = \sqrt{\frac{1}{N} \sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} \text{dist}^2(\mathbf{x}, F_k)}, \tag{4}$$

where F_k is the OLS d-flat approximating C_k (obtained by Principal Component Analysis (PCA)), and $\text{dist}(\mathbf{x}, F_k)$ denotes the orthogonal distance from $\mathbf{x}$ to F_k. In situations where we know the true membership of the data points, we also compute the percentage of misclassified points. That is,

$$e_{\%} = \frac{\#\text{ of misclassified points}}{N} \cdot 100\%. \tag{5}$$

3 The SCC Algorithm

We first introduce several numerical techniques (in Sect. 3.1) to make the TSCC algorithm practical and then form the SCC algorithm (in Sect. 3.2). We next analyze its complexity in terms of both storage and running time (in Sect. 3.3), and finally propose two more strategies: one for isolating outliers (in Sect. 3.4), and the other for segmenting flats of mixed dimensions (in Sect. 3.5).

3.1 The Novel Methods of SCC

3.1.1 Iterative Sampling

The TSCC algorithm is not applicable in practice for two reasons: First, the amount of space for storing the affinity matrix $\mathbf{A} \in \mathbb{R}^{N \times N^{d+1}}$ can be huge ($O(N^{d+2})$); Second, full computation of $\mathbf{A}$ and multiplication of this large matrix and its transpose (to produce $\mathbf{W}$) can be computationally prohibitive. One solution might be to use uniform sampling, i.e., randomly select and compute a small subset of the columns of $\mathbf{A}$, to produce an estimate of $\mathbf{W}$ (Drineas et al. 2006; Govindu 2005),[1] which is stated below.

Denoting by $\mathbf{A}(:, j)$ the j^{th} column of $\mathbf{A}$, we write $\mathbf{W}$ in the following way:

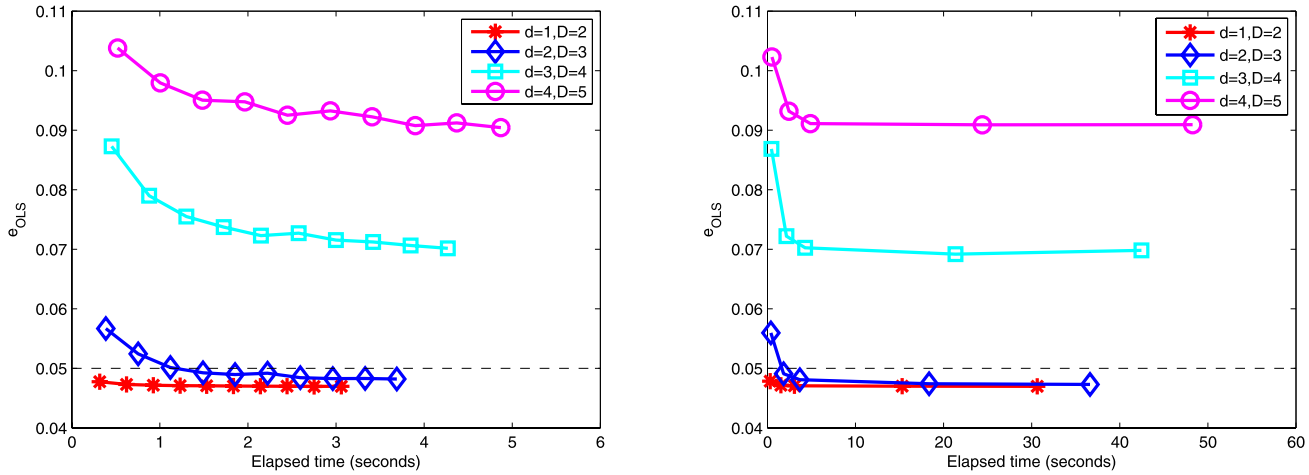
$$\mathbf{W} = \sum_{j=1}^{N^{d+1}} \mathbf{A}(:, j) \cdot \mathbf{A}(:, j)'. \tag{6}$$

Consequently, $\mathbf{W}$ is a sum of N^{d+1} rank-1 matrices. Let $j_1, \ldots, j_c$ be c integers that are randomly selected between 1 and N^{d+1}. Then $\mathbf{W}$ can be approximated as follows (Drineas et al. 2006):[2]

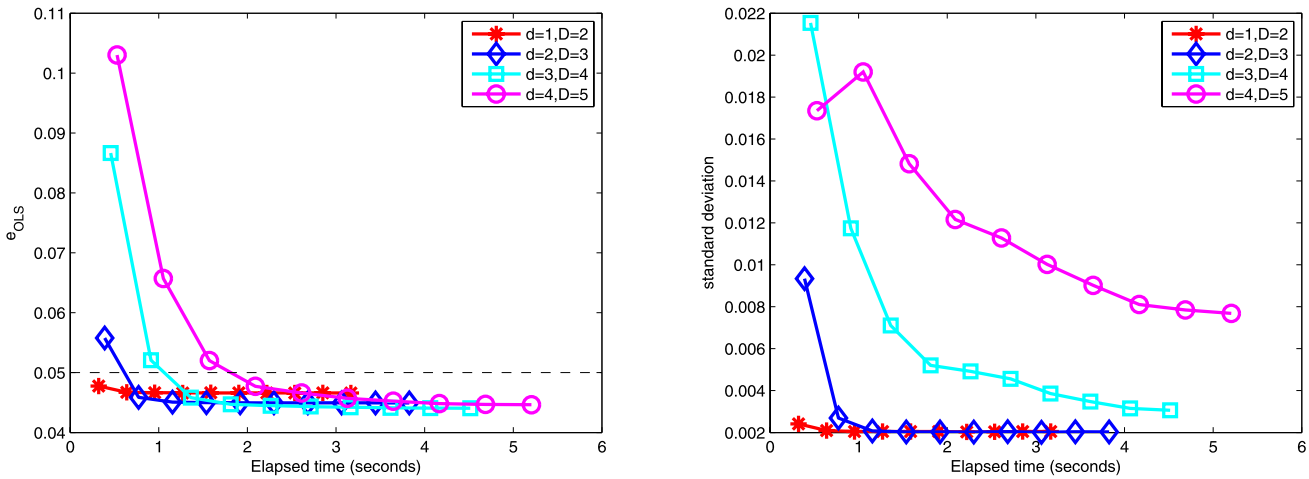
$$\mathbf{W} \approx \sum_{t=1}^{c} \mathbf{A}(:, j_t) \cdot \mathbf{A}(:, j_t)'. \tag{7}$$

[1] In Drineas et al. (2006) a more accurate sampling scheme according to the magnitudes of the columns is also suggested. Nevertheless, since we do not have the full affinity matrix $\mathbf{A}$, this technique can not be applied in our setting.

[2] More precisely, a scaling constant needs to be used in front of the sum in order to have the right magnitude (see Drineas et al. 2006, Sect. 4). However, since we are only interested in the eigen-structure of $\mathbf{W}$, this constant is omitted.



(**a**) Uniform Sampling: The averaged errors obtained using different choices of c. On each curve a symbol represents a distinct value of c. *Left*: c is taken to be $N, 2N, \ldots, 10N$ respectively; *Right*: $c = N, 5N, 10N, 50N, 100N$



(**b**) Iterative Sampling: The mean (*left*) and standard deviation (*right*) of the errors obtained in the initial step (uniform sampling) and the first 9 updates using iterative sampling with $c = N = 100 \cdot K$ always fixed

Fig. 1 Plots of the errors (e_{OLS}) using different sampling strategies against time. In each experiment we randomly generate $K = 3$ d-dimensional linear subspaces in $\mathbb{R}^D$. Each subspace contains 100 points, so $N = 100 \cdot K$. The model error is 0.05 in all situations (indicated by the *dashed lines*). We repeat this experiment 500 times (for each fixed pair (d, D)) in order to compute an average of e_{OLS}

In practice, in order to have at most quadratic complexity, we expect the maximum possible c to be an absolute constant or a small number times N, resulting in $c/N^{d+1} \leq O(N^{-d})$. We thus conclude that uniform sampling (maintaining quadratic complexity) is almost surely not able to capture the column space of $\mathbf{A}$ when N is large and d is moderate. Indeed, this is demonstrated in Fig. 1(a): In the two cases where $d > 2$, the error e_{OLS} does not get close to the model error even with $c = 100 \cdot N$. This illustrates a fundamental limitation of uniform sampling. In the following we explain our strategy to resolve this issue.

We note that each column j of $\mathbf{A}$ uniquely corresponds to an ordered list of $d + 1$ points $(\mathbf{x}_{j_1}, \mathbf{x}_{j_2}, \ldots, \mathbf{x}_{j_{d+1}})$ in the data, and moreover, repeated points lead to a zero column (see (3)). Thus, we will select only tuples of $d + 1$ distinct points in X when sampling columns of $\mathbf{A}$.

We say that an n-tuple of points is *pure* if these n points are in the same underlying cluster, and that it is *mixed* otherwise. Similarly, a column of the matrix $\mathbf{A}$ is said to be pure if it corresponds to a pure $(d + 1)$-tuple, and mixed otherwise. We use these two categories of columns of $\mathbf{A}$ to explain our sampling strategy.

In the ideal case (see Sect. 2.3), any mixed column of $\mathbf{A}$ is identically zero and thus makes no contribution to computing the matrix $\mathbf{W}$. On the other hand, the pure columns lead to a block diagonal structure of $\mathbf{W}$, which guarantees a

perfect segmentation (Chen and Lerman 2007, Sect. 4.1). In practice the mixed columns are typically not all zero. Since the percentage of these columns in **A** is high, the matrix **W** loses the desired block diagonal structure. If we only use the pure columns of **A**, then we can expect **W** to be nearly block diagonal.

The iterative sampling scheme is motivated by the above observations and works as follows. We fix c to be some constant, e.g., $c = 100 \cdot K$. Initially, c columns of **A** are randomly selected and computed so as to produce **W**, and then an initial segmentation of X into K clusters is obtained with this **W** (we call this initial step *the zeroth iteration*). We then re-sample c columns of **A** by selecting c/K columns from within each of the K initially found clusters, or from the points within a small strip around the OLS d-flat of each such cluster, and obtain K newer clusters. In order to achieve the best segmentation, one can iterate this process a few times, as the newer clusters are expected to be closer to the underlying clusters.

We demonstrate the strength of this sampling strategy by repeating the experiments in Fig. 1a, but with iterative sampling replacing uniform sampling. Due to the randomness of sampling, we compute both the mean and the standard deviation of the errors e_{OLS} in the 500 experiments in each of the intermediate steps of iterative sampling (see Fig. 1b). In all cases, the mean drops rapidly below the model error when iterating, and the standard deviation also decays quickly.

We remark that as d increases, we should also use larger c in the zeroth iteration in order to capture "enough" pure columns. Indeed, in order to have (on average) c_0 pure columns sampled from each underlying cluster in the zeroth iteration, we need to have $c \approx c_0 \cdot K^{d+2}$. Afterwards, we may still reduce c to a constant multiple of K in the subsequent iterations. We plan to study more carefully the required magnitudes of c (for the zeroth iteration and the subsequent iterations respectively) to ensure convergence. When the theoretical value of c is unrealistically large, we can sample columns from the output of other d-flats clustering algorithms (e.g., K-Subspaces) to initialize SCC.

3.1.2 Estimation of the Tuning Parameter σ

The choice of the tuning parameter σ is crucial to the performance of any algorithm involving Gaussian-kernel affinities. However, selecting its optimal value is not an easy task, and is insufficiently investigated in the literature. Common practice is to manually select a small set of values and choose the one that works the best (e.g., Ng et al. 2002). Since the optimal value of σ should depend on the scale of the data, subjective choices may work poorly (see Fig. 2). We develop an automatic scheme to infer the optimal value of σ (or an interval containing it) from the data itself.

We start by assuming that all curvatures are computed (which is unrealistic when $d > 1$). In this case, we estimate the correct choice of σ, starting with the clean case and then corrupting it by noise. We follow by examining the practical setting of c sampled columns, i.e., when only a fraction of the curvatures are computed.

In the clean case, the polar curvatures of all pure $(d+2)$-tuples are zero. In contrast, (almost) all mixed $(d+2)$-tuples have positive curvatures.[3] By taking a sufficiently small $\sigma > 0$ the resulting affinity tensor can closely approximate the perfect tensor (defined in Sect. 2.3), thus an accurate segmentation is guaranteed. When the data is corrupted with moderate noise, we still expect the curvatures of most pure $(d+2)$-tuples to be small, and those of most mixed $(d+2)$-tuples to be large. The optimal value of σ, σ_{opt}, is the maximum of the small curvatures corresponding to pure tuples (up to a scaling constant). Indeed, transforming the curvatures by $\exp(-(\cdot)^2/(2\sigma_{\mathrm{opt}}^2))$ will produce affinities that are close to zero (for mixed tuples) and one (for pure tuples). In other words, this transformation serves like a "low-pass filter": It "passes" smaller curvatures by producing large affinities toward one, and "blocks" bigger curvatures toward zero.

Therefore, in the case of small within-cluster curvatures and large between-cluster curvatures, one can compute all the curvatures, have them sorted in an increasing order, estimate the number of small curvatures corresponding to pure tuples, and take as σ_{opt} the curvature value at that particular index in the sorted vector. The key step is determining the index of that curvature value. For this reason we refer to our approach as *index estimation*.

We next obtain this index in two cases. We denote by $P(n,r)$ the number of permutations of size r from n available elements, where $n \geq r \geq 1$ are integers. For any $1 \leq j \leq K$, let N_j denote the size of the j^{th} underlying cluster. First of all, we suppose that these N_j are given. Then the proportion of pure $(d+2)$-tuples to all $(d+2)$-tuples equals:

$$\gamma = \frac{\sum_{1 \leq j \leq K} P(N_j, d+2)}{P(N, d+2)} \approx \sum_{j=1}^{K} \left(\frac{N_j}{N} \right)^{d+2}. \tag{8}$$

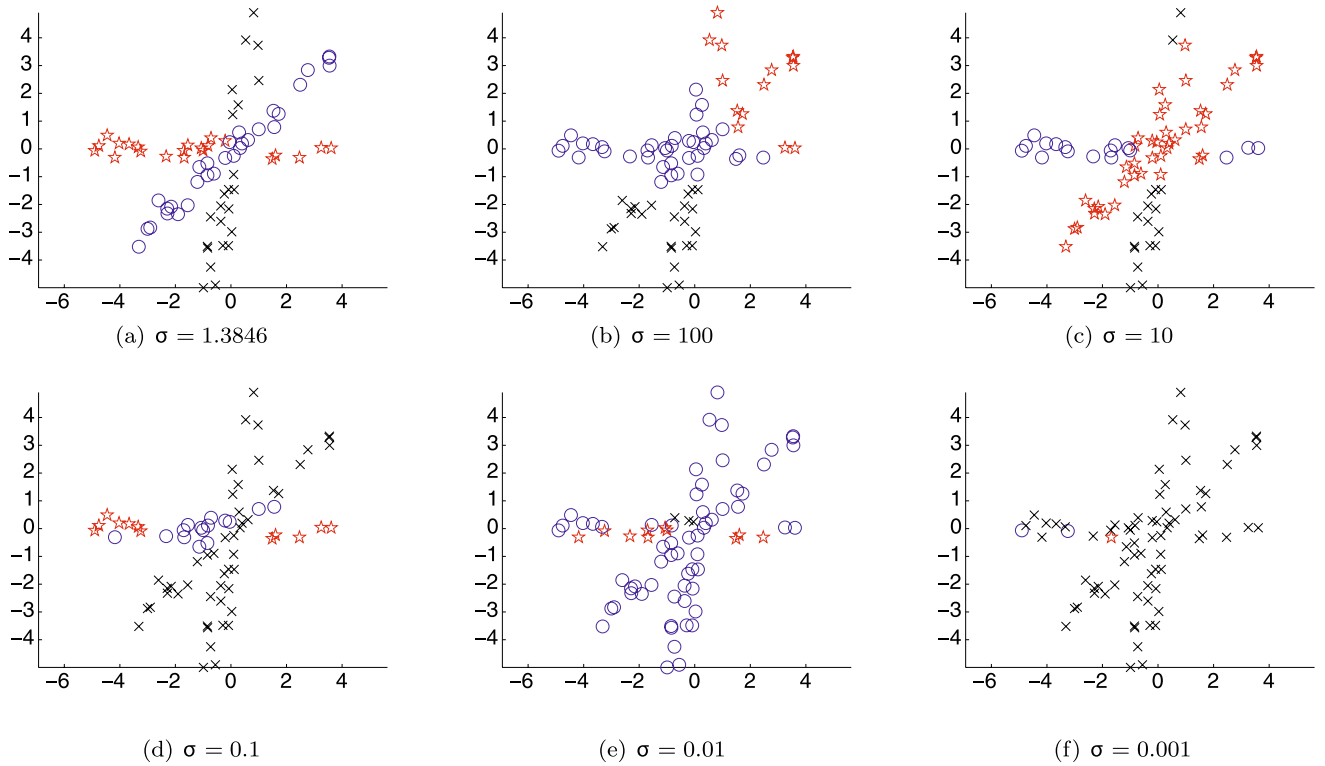
That is, the curvature value at the index of $\gamma \cdot P(N, d+2)$ can be used as the best estimate for the optimal σ. Second, when N_j are unknown, we work out the absolute minimum[4]

[3] When a mixed $(d+2)$-tuple happen to be lying on a d-flat, the polar curvature will be correspondingly zero. However, such mixed tuples should be rare.

[4] The absolute minimum can be obtained by solving a constrained optimization problem:

$$\min_{\gamma_1, \ldots, \gamma_K \geq 0} \sum_{j=1}^{K} \gamma_j^{d+2} \quad \text{subject to} \quad \sum_{j=1}^{K} \gamma_j = 1.$$

The minimum is attained when $\gamma_j = 1/K,\ j = 1, \ldots, K$.



(a) σ = 1.3846 (b) σ = 100 (c) σ = 10

(d) σ = 0.1 (e) σ = 0.01 (f) σ = 0.001

Fig. 2 Segmentation results with different choices of σ. The value 1.3846 is inferred from the data using our strategy (explained in Example 3.1); the other values are manually selected

of the last quantity in (8) and use it as a lower bound for the fraction γ:

$$\gamma \gtrapprox K \cdot \left(\frac{1}{K}\right)^{d+2} = \frac{1}{K^{d+1}}. \tag{9}$$

We note that if all N_j are equal to N/K, then this lower bound coincides with its tighter estimate provided in (8). The following example demonstrates this strategy.

Example 3.1 We take the data in Fig. 2 which consist of three lines in $\mathbb{R}^2$, each having 25 points. This data set has a relatively small size, so we are able to compute all the polar curvatures. We apply (8) (or (9)) and obtain that $\gamma \approx 1/9$. Thus, we use the $1/9 \cdot P(75, 2) = 617^{\text{th}}$ smallest curvature as the optimal value of the tuning parameter: $\sigma = 1.3846$.

We now go to our practical setting (Sect. 3.1.1) where we iteratively sample only c columns of $\mathbf{A}$ and thus do not have all the curvatures. We assume convergence of the iterative sampling so that the proportion of pure columns (in the c sampled columns) increases with the iterations. Consequently, we obtain a lower bound for σ from the zeroth iteration, and an upper bound from the last iteration.

In the zeroth iteration (uniform sampling) c columns of $\mathbf{A}$ are randomly selected. We expect to have the same lower bound as in (9) for the proportion of pure $(d+2)$-tuples in these c columns. We note that there are exactly $N-d-1$ tuples of $d+2$ distinct points in each of these c columns. Denoting by $\mathbf{c}$ the vector of the $(N-d-1)\cdot c$ corresponding curvatures sorted in an increasing order, we write a lower bound for σ as follows:

$$\sigma_{\min} = \mathbf{c}\left((N-d-1)\cdot c/K^{d+1}\right). \tag{10}$$

In the last iteration (when the scheme converges to finding the true clusters), c/K columns are sampled from each of the K underlying clusters, thus all the c columns are pure. In this case, the number of pure $(d+2)$-tuples in the c columns attains the following maximum possible value:

$$\sum_{j=1}^{K}(N_j - d - 1)\cdot\frac{c}{K} = N\cdot c/K - (d+1)\cdot c.$$

Therefore, we have the following upper bound for σ:

$$\sigma_{\max} = \mathbf{c}((N/K - d - 1)\cdot c). \tag{11}$$

We present two practical ways of searching the interval $[\sigma_{\min}, \sigma_{\max}]$ for the optimal value of σ. First, one can start with the upper bound $\sigma_{\max}$ and divide it by a constant (e.g., $\sqrt{2}$) each time until it falls below the lower bound $\sigma_{\min}$. Second, one can search by the index of the vector $\mathbf{c}$, i.e., choose

the optimal value from a subset of $\mathbf{c}$:

$$\{\mathbf{c}\left(N \cdot c/K^q\right) \mid q = 1, \ldots, d+1\}. \tag{12}$$

We will use the second strategy (which only tests $d+1$ values) in the SCC algorithm, as we find in experiments that it works sufficiently well. To further improve efficiency, we can gradually raise the lower bound (i.e., $\sigma_{\min}$) in the subsequent iterations.

3.1.3 Initialization of K-Means

The clustering step in the TSCC algorithm applies K-means to the rows of $\mathbf{U}$. In the ideal case, these rows coincide with K mutually orthogonal vectors (the "seeds") in $\mathbb{R}^K$ (Chen and Lerman 2007, Proposition 4.1); in the case of noise, the rows of $\mathbf{U}$ correspond to more than K points that originate from those seeds and possibly overlap in between. See Fig. 3 for an illustration. We locate these seeds by maximizing the variance among all possible combinations of K rows of $\mathbf{U}$, and then use them to initialize K-means.

Formally, the indices of these seeds can be found by solving the following optimization problem:

$$\{s_1, \ldots, s_K\} = \arg \max_{1 \le n_1 < \cdots < n_K \le N} \sum_{i=1}^{K} \left\| \mathbf{U}(n_i, :) - \frac{1}{K} \cdot \sum_{j=1}^{K} \mathbf{U}(n_j, :) \right\|^2. \tag{13}$$

With a little algebra we obtain an equivalent representation:

$$\{s_1, \ldots, s_K\} = \arg \max_{1 \le n_1 < \cdots < n_K \le N} \sum_{1 \le i < j \le K} \left\| \mathbf{U}(n_i, :) - \mathbf{U}(n_j, :) \right\|^2. \tag{14}$$

We thus apply an inductive scheme (via (14)) to solve the above maximization problem. The first index s_1 is chosen to be that of the row farthest from the center of all N rows. That is,

$$s_1 = \arg \max_{1 \le n \le N} \left\| \mathbf{U}(n, :) - \frac{1}{N} \sum_{i=1}^{N} \mathbf{U}(i, :) \right\|. \tag{15}$$

Suppose now that $1 \le k < K$ seeds have been chosen, then the index of the $(k+1)^{\text{st}}$ seed is determined by

$$s_{k+1} = \arg \max_{\substack{1 \le n \le N \\ n \ne s_1, \ldots, s_k}} \sum_{i=1}^{k} \left\| \mathbf{U}(s_i, :) - \mathbf{U}(n, :) \right\|^2. \tag{16}$$

Remark 3.2 When the rows of $\mathbf{U}$ are normalized to have unit length, we still use the criterion in (14), which correspondingly reduces to minimizing the sum of inner products among all possible combinations of K rows of $\mathbf{U}$. In this case, our strategy, i.e., (15) and (16), is still different from that of Ng et al. (2002).

3.2 The SCC Algorithm

We combine together the theoretical algorithm (Algorithm 1) and all the techniques introduced in Sect. 3.1 to form a comprehensive Spectral Curvature Clustering (SCC) algorithm for practical use (Algorithm 2).

Algorithm 2 Spectral Curvature Clustering (SCC)

Input: Data set X, intrinsic dimension d, number of d-flats K, number of sampled columns c (default $= 100 \cdot K$).

Output: K disjoint clusters $\mathrm{C}_1, \ldots, \mathrm{C}_K$ and error e_{OLS}.

Steps:

1: Sample randomly c subsets of X, each containing exactly $d+1$ distinct points.

2: Compute the polar curvature of any subset and each of the rest of $N-d-1$ points in X by (2), and sort increasingly these $(N-d-1) \cdot c$ curvatures into a vector $\mathbf{c}$.

3: **for** $q = 1 : d+1$ **do**

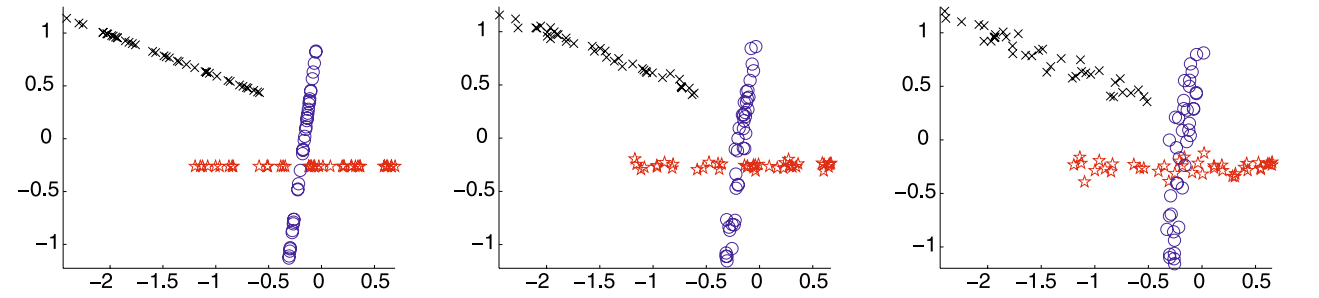
- Use (3) together with $\sigma = \mathbf{c}\left(N \cdot c/K^q\right)$ to compute the c selected columns of $\mathbf{A}$. Form a matrix $\mathbf{A}_c \in \mathbb{R}^{N \times c}$ using these c columns.
- Compute $\mathbf{D} = \mathrm{diag}\{\mathbf{A}_c \cdot (\mathbf{A}_c' \cdot \mathbf{1})\}$ and use it to normalize $\mathbf{A}_c$: $\mathbf{A}_c^* = \mathbf{D}^{-1/2} \cdot \mathbf{A}_c$.
- Form the matrix $\mathbf{U}$ whose columns are the top K left singular vectors of $\mathbf{A}_c^*$.
- Apply K-means, initialized using (15) and (16), to the rows of $\mathbf{U}$ (possibly normalized to have unit length), and separate them into K clusters.
- Use these detected clusters to group the points of X into K subsets, and compute the corresponding error e_{OLS} using (4).

end for

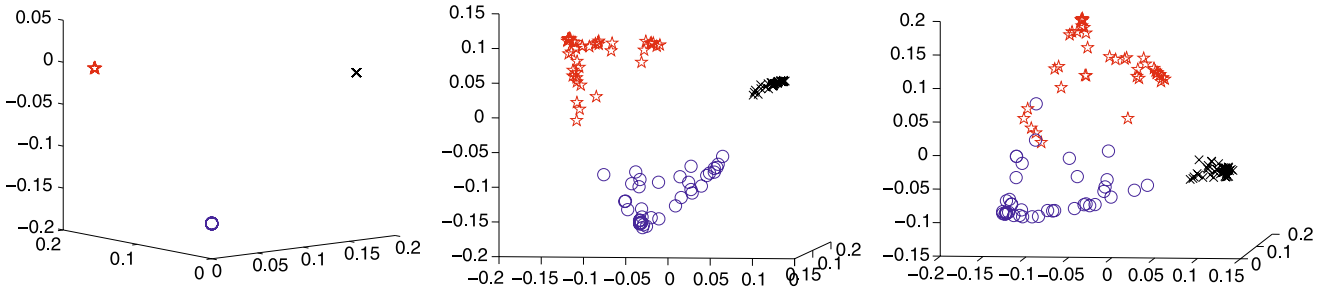
Record the K subsets $\mathrm{C}_1, \ldots, \mathrm{C}_K$ of X that correspond to the smallest error e_{OLS} in the above loop.

4: Sample c/K $(d+1)$-tuples from each C_j, $1 \le j \le K$ found above (or the points within a small strip around each of their OLS d-flats), and repeat Steps 2 and 3 to find K newer clusters. Iterate until convergence to obtain a best segmentation result.

Remark 3.3 (Linear SCC Algorithm (LSCC)) If all the subspaces are known to be linear, then one can take any $d+1$



(a) Data and clusters



(b) The rows of **U**

Fig. 3 Three data sets of the same model but with increasing levels of noise, and their images in the embedded space

points together with the origin to compute a curvature. This simplifies the SCC algorithm by producing a smaller affinity tensor (of order $d+1$). Moreover, it can improve the segmentation result for the following two reasons: (1) The new matrix $\mathbf{A}$ has less columns than before by one order of N, so the same number of sampled columns can be a better representative of the column space of $\mathbf{A}$; (2) For $d+1$ points and the origin, a small curvature always implies that the $d+1$ points are close to being on some underlying linear subspace. This excludes the unfavorable small curvatures for $d+2$ points around an affine subspace (which is the case for SCC).

3.3 Complexity of the SCC Algorithm

The implementation of the SCC algorithm is mainly through standard matrix operations, such as elementwise manipulation, matrix-vector multiplication, Singular Value Decomposition (SVD), etc. Consequently, the complexity of SCC is completely determined by the sizes of the matrices used in the algorithm and the types of operations between them.

The storage requirement of the algorithm is $O(N\cdot(D+c))$. Indeed, the biggest matrices are X (when considered as a matrix), whose size is $N\times D$, and $\mathbf{A}_c, \mathbf{A}_c^*$ (defined in Algorithm 2), which have size $N\times c$. In order to estimate the running time, we first note that it takes $O((d+1)\cdot D\cdot N\cdot c))$ time to compute $\mathbf{A}_c$ by using matrix manipulations (see code at the supplementary webpage). Also, it takes $O(N\cdot c)$ time to compute $\mathbf{D}$, $\widetilde{\mathbf{A}}_c$, and $O((N+c)\cdot K^2)$ time to calculate $\mathbf{U}$ by fast SVD algorithms (e.g., Brand 2003). Thus, each iteration takes $O((d+1)^2\cdot D\cdot N\cdot c)$ time (the computation is repeated $d+1$ times in Step 3 of Algorithm 2). Let n_s denote the number of sampling iterations performed. We then obtain that the total running time of the SCC algorithm is $O(n_s\cdot(d+1)^2\cdot D\cdot N\cdot c)$.

3.4 Outliers Detection

We detect outliers according to the degrees of the data points, i.e., the diagonal elements of the matrix $\mathbf{D}$ in Algorithm 2. We assume that the percentage of outliers is known. In each sampling iteration, after the degrees $\mathbf{D}$ have been computed, we isolate the percentage of points with the smallest degrees as intermediate outliers, and remove the corresponding rows from the matrix $\mathbf{A}_c$. We then recompute $\mathbf{D}$ and follow the subsequent steps of SCC to obtain K clusters. In the next iteration, we will sample c/K columns only from each of the previously detected clusters to form the matrix $\mathbf{A}_c\in\mathbb{R}^{N\times c}$ (thus excluding the previous outliers). Those outliers isolated in the final sampling iteration will be the ultimate outliers.

To evaluate the performance of this outliers detection strategy associated with SCC, we plot in Fig. 4 a *Receiver*

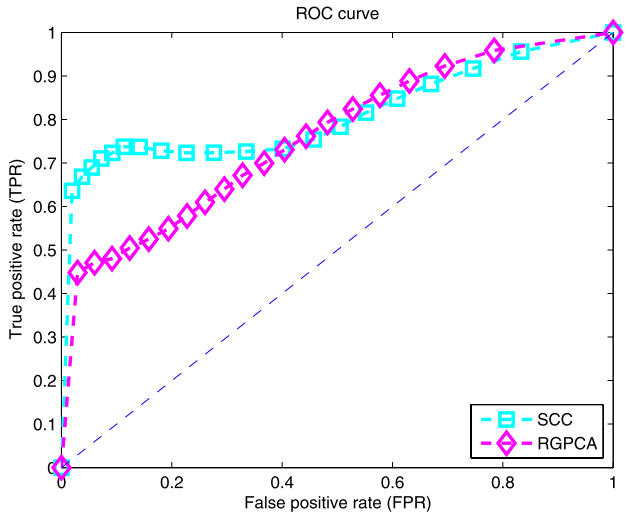


Fig. 4 ROC curves corresponding to SCC and RGPCA. We randomly generate $K = 3$ linear lines in $\mathbb{R}^2$, and sample 100 points from each line. The samples are then corrupted with 5% Gaussian noise and further contaminated with some percentage of outliers. The percentages used are 5%, 10%, 15%, ..., 95% respectively, as indicated by the symbols on each curve. For each fixed percentage, 500 experiments are repeated in order to compute an average for each of the two rates TPR and FPR

Operating Characteristic (*ROC*) *curve* in the case of lines contaminated with outliers. An ROC curve is the plot of the *true positive rates* (TPR) against the *false positive rates* (FPR). The TPR is the percentage of correctly detected outliers, while the FPR is the percentage of the data points in the stable distribution which are falsely detected as outliers. A large area under the ROC curve is indication of good performance in outliers detection for a wide range of FPRs. The area of the region under the ROC curve corresponding to SCC is 0.8105. In comparison, the Robust GPCA algorithm (RGPCA) (Ma et al. 2008) has an area of 0.7613 under its ROC curve. The figure also emphasizes the fact that SCC has a better performance than RGPCA at low FPRs which are practically more important.

3.5 Mixed Dimensions

The SCC algorithm is formulated in the setting of data sampled from flats of the same dimension d. In fact, it can be easily adapted to cluster flats of mixed dimensions, i.e., when the dimensions $d_1, d_2, \ldots, d_K$ are not necessarily the same.

Our strategy is to use the maximum of the dimensions

$$d_{\max} = \max_{1 \le j \le K} d_j, \tag{17}$$

and apply SCC to segment K $d_{\max}$-dimensional flats. We find in experiments that this technique results in small segmentation errors $e_{\%}$. At this stage we cannot compute e_{OLS} since we do not know the intrinsic dimensions of the detected clusters. We will try to resolve this issue in later research.

4 Numerical Experiments

4.1 Simulations

We compare the SCC algorithm (and also LSCC when applicable) with other competing methods on a number of artificial data sets in the setting of hybrid linear modeling.

The three methods that we compare with are the Mixtures of PPCA algorithm (MoPPCA) (Tipping and Bishop 1999), the K-Subspaces algorithm (KS) (Ho et al. 2003), and the GPCA algorithm with voting (GPCA) (Ma et al. 2008). We use the Matlab codes of the GPCA algorithm at http://perception.csl.uiuc.edu/gpca/. We also borrow from that website the Matlab code that generates various data sets. MoPPCA and KS were implemented by Stefan Atev and ourselves respectively (see codes at the supplementary webpage). These two methods are always initialized with a random guess of the membership of the data points. Due to the randomness in the initialization, multiple restarts are used and the best segmentation result is recorded.

The three multi-way clustering algorithms (Agarwal et al. 2005; Govindu 2005; Shashua et al. 2006) seem highly related and should have been included for comparison. However, they mainly focus on how to process a given affinity tensor; many practical issues are not fully discussed in the context of hybrid linear modeling, and are also missing from their implementation. In fact, we have compared with Govindu (2005) regarding random sampling and choices of the tuning parameter σ (see Figs. 1 and 2).

In the following we conduct experiments in the cases of linear/affine subspaces of the same dimension/mixed dimensions to compare the performance of the four algorithms, namely MoPPCA, KS, GPCA, and SCC. The simulations were performed on a compute server with two dual-core AMD Opteron 64-bit 280 processors (2.4 GHz) with 8 GB of RAM. We remark that when applying SCC (Algorithm 2) we fix $c = 100 \cdot K$, and do not normalize the rows of $\mathbf{U}$.

We first randomly generate K *linear* subspaces of a fixed dimension d in some Euclidean space $\mathbb{R}^D$, which we write $d^K \in \mathbb{R}^D$ for short. We follow Ma et al. (2008) to restrict the angles between these subspaces to be at least 30 degrees in order to ensure enough separation. Also, the diameter of each subspace is fixed to be 1. We then randomly draw 100 samples from each of the subspaces, and corrupt them with 5% Gaussian noise. We apply the four algorithms to the data and record both types of errors e_{OLS} and $e_{\%}$ as well as the computation time t. This experiment is repeated 500 times and the averaged errors and time are shown in Table 1. In all

Table 1 The two types of errors $e_{\text{OLS}}, e_{\%}$ and computation time t (in seconds) of the four methods when clustering *linear* subspaces. The number of subspaces K and the intrinsic dimension d are given to all algorithms. The MoPPCA and KS algorithms are randomly initialized. Ten restarts are used for each of them, and the smallest error is used

	$1^6 \in \mathbb{R}^3$			$2^4 \in \mathbb{R}^3$			$3^3 \in \mathbb{R}^4$			$4^3 \in \mathbb{R}^6$		
	e_{OLS}	$e_{\%}$	t	e_{OLS}	$e_{\%}$	t	e_{OLS}	$e_{\%}$	t	e_{OLS}	$e_{\%}$	t
MoPPCA	.048	8.3%	0.8	.042	19.2%	0.7	.043	16.8%	0.4	.048	3.2%	0.4
KS	.048	8.0%	0.4	.043	19.5%	0.2	.043	16.3%	0.2	.048	3.1%	0.2
LSCC	.048	8.6%	1.6	.043	19.8%	1.8	.044	17.3%	1.5	.048	3.4%	1.8
SCC	.050	9.9%	3.1	.048	23.1%	3.0	.044	18.2%	2.0	.048	3.6%	2.1
GPCA	.184	39.8%	3.7	.088	39.5%	1.5	.077	32.5%	1.3	.126	31.7%	3.1

Table 2 The two types of errors $e_{\text{OLS}}, e_{\%}$ and computation time t (in seconds) of the four algorithms when clustering *affine* subspaces. The number of subspaces K and the intrinsic dimension d are given to all algorithms. The MoPPCA and KS algorithms are randomly initialized. Ten restarts are used for each of them, and the smallest error is used

	$1^4 \in \mathbb{R}^2$			$2^3 \in \mathbb{R}^3$			$3^3 \in \mathbb{R}^4$			$4^3 \in \mathbb{R}^5$		
	e_{OLS}	$e_{\%}$	t	e_{OLS}	$e_{\%}$	t	e_{OLS}	$e_{\%}$	t	e_{OLS}	$e_{\%}$	t
GPCA	.174	29.1%	1.3	.116	20.1%	1.0	.128	25.2%	1.2	.138	30.2%	1.5
MoPPCA	.110	35.4%	0.7	.115	47.6%	0.6	.101	49.6%	0.7	.089	49.0%	0.9
KS	.089	25.5%	0.2	.113	45.4%	0.1	.101	49.2%	0.2	.090	49.3%	0.2
SCC	.049	4.2%	1.3	.049	2.8%	1.0	.048	2.0%	1.4	.048	1.4%	2.1

Table 3 The percentage of misclassified points $e_{\%}$ and elapsed time t (in seconds) by all algorithms when clustering hybrid data sampled from linear/affine subspaces of *mixed* dimensions. The dimensions of the subspaces are given to all the algorithms. The MoPPCA and K-Subspaces algorithms are randomly initialized. Ten restarts are used for each of them, and the smallest error is used

	$(1,2,2) \in \mathbb{R}^3$				$(1,1,2) \in \mathbb{R}^3$				$(1,1,2,2) \in \mathbb{R}^3$				$(1,2,3) \in \mathbb{R}^4$			
	Linear		Affine		Linear		Affine		Linear		Affine		Linear		Affine	
	$e_{\%}$	t	$e_{\%}$	t	$e_{\%}$	t	$e_{\%}$	t	$e_{\%}$	t	$e_{\%}$	t	$e_{\%}$	t	$e_{\%}$	t
KS	10.6%	0.1	34.1%	0.1	11.2%	0.1	26.9%	0.1	21.8%	0.2	36.6%	0.2	19.5%	0.1	38.6%	0.1
MoPPCA	8.0%	0.4	41.4%	0.6	24.0%	0.4	37.6%	0.6	20.4%	0.8	44.0%	1.0	24.0%	0.6	31.8%	0.7
GPCA	7.3%	2.2	11.7%	2.5	17.8%	2.1	18.1%	1.9	25.2%	3.5	24.7%	3.6	13.2%	6.4	17.4%	4.5
SCC	7.2%	1.2	1.0%	0.9	9.2%	1.1	0.5%	0.9	18.6%	2.2	1.4%	1.6	8.4%	1.6	0.3%	1.4
LSCC	6.1%	0.7			7.1%	0.8			10.8%	1.3			6.6%	1.5		

the four scenarios, MoPPCA, KS, SCC and LSCC have comparable performance, but they all outperform GPCA at $1–10^{-7}$ confidence level using paired t-tests.

We next compare the SCC algorithm with the other methods on clustering *affine* subspaces. We generate affine subspaces with the same controlling parameters as in the linear case. We remark that the software we are using (borrowed from http://perception.csl.uiuc.edu/gpca/) tries to avoid intersection of these affine subspaces, or more precisely, of the sampled clusters. We note that, since SCC does not distinguish between linear and affine subspaces, its performance in the case of intersecting affine subspaces can be reflected in Table 1 (where we have intersecting linear subspaces). The two types of errors due to all four methods and their computation time are recorded in Table 2. The results of paired t-tests between SCC and the other three methods show that SCC performs better at $1–10^{-7}$ confidence level in terms of both errors.

We finally compare all the algorithms on clustering linear/affine subspaces of *mixed* dimensions in order to further evaluate their performance. We follow the notation in Ma et al. (2008) to denote data sampled from subspaces of mixed dimensions by $(d_1, \ldots, d_K) \in \mathbb{R}^D$. All the parameters used in generating the data are the same as above, except that the noise level becomes 3%. Table 3 shows the percentage of misclassified points (i.e., $e_{\%}$) and elapsed time by each

of the four algorithms in eight scenarios. Without determining the true dimensions, the LSCC (resp. SCC) algorithm in the case of linear (resp. affine) subspaces still exhibits better performance in terms of $e_{\%}$ than its competitors at $1–10^{-7}$ confidence level (using paired t-tests).

4.2 Applications

Hybrid linear modeling has broad applications in many areas, such as computer vision, image processing, pattern recognition, and system identification. We exemplify below the application of the SCC algorithm to a few real-world problems that are studied in Vidal et al. (2005) and Ma et al. (2008).

4.2.1 Motion Segmentation under Affine Camera Models

Suppose that a video sequence consists of F frames of images of several objects that are moving independently against the background, and that N feature points $\mathbf{y}_1, \ldots, \mathbf{y}_N \in \mathbb{R}^3$ are detected on the objects and the background. Let $\mathbf{z}_{ij} \in \mathbb{R}^2$ be the coordinates of the feature point $\mathbf{y}_j$ in the i^{th} image frame for every $1 \leq i \leq F$ and $1 \leq j \leq N$. Then $\mathbf{z}_j = [\mathbf{z}'_{1j}\, \mathbf{z}'_{2j}\, \ldots\, \mathbf{z}'_{Fj}]' \in \mathbb{R}^{2F}$ represents the trajectory of the j^{th} feature point across the F frames. The problem is how to separate these trajectory vectors $\mathbf{z}_1, \ldots, \mathbf{z}_N$ into independent motions undertaken by those objects and the background.

It has been shown (e.g., in Ma et al. 2008) that, under affine camera models and with some mild conditions, the trajectory vectors corresponding to different moving objects and the background across the F image frames live in distinct linear subspaces of dimension at most four in $\mathbb{R}^{2F}$, or affine subspaces of dimension at most three within those linear subspaces.

We borrow the data from Sugaya and Kanatani (2004), which is also used in Ma et al. (2008). This data consist of two outdoor sequences taken by a moving camera tracking a car moving in front of a parking lot and a building (Sequences A and B), and one indoor sequence taken by a moving camera tracking a person moving his head (Sequence C), as shown in Sugaya and Kanatani (2004), Fig. 7.

Following the above theory, we first apply SCC (Algorithm 2) as well as LSCC to segment two 4-dimensional linear subspaces in $\mathbb{R}^{2F}$ for each of the three sequences. We also apply SCC to each sequence and segment 3-dimensional affine subspaces in $\mathbb{R}^{2F}$. In all these cases, SCC obtains 100% accuracy. In contrast, GPCA cannot be applied directly to the original trajectories in Sequences A and C, as it is computationally too expensive to find all the normal vectors of these low dimensional linear subspaces within a high dimensional ambient space. Even in Sequence B where we could apply GPCA (since $2F = 34$ is moderate), it produces varying errors, which are sometimes nearly 40% (see Table 4).

Table 4 Percentage of misclassified points $e_{\%}$ by SCC and GPCA respectively using different combinations (d, D). Here d is the dimension of the subspaces, and D is the ambient dimension. Both algorithms are without post-optimization. In the table below N/A represents *Not Applicable*, while VE is short for *Varying Errors*

Sequence		A	B	C
Number of points N		136	63	73
Number of frames F		30	17	100
SCC/LSCC	$d = 4, D = 2F$	0%	0%	0%
SCC	$d = 3, D = 2F$	0%	0%	0%
PCA+SCC	$d = 3, D = 4$	0%	0%	0%
GPCA	$d = 3/4, D = 2F$	N/A	VE	N/A
SVD+GPCA	$d = 4, D = 5$	0%	0%	40%

To further evaluate the performance of the two algorithms, we have also applied GPCA and SCC to the three sequences after reducing the ambient dimensions. We first project the trajectories onto a 5-dimensional space by direct SVD (to maintain the linear structure), and apply GPCA to segment 4-dimensional linear subspaces in $\mathbb{R}^5$ as suggested in Ma et al. (2008), but without post-optimization by KS. A segmentation error of 40% is obtained for Sequence C (see Table 4). The equivalent way of applying SCC is to first project the data onto the first four principal components by PCA, and then segment 3-dimensional affine subspaces in $\mathbb{R}^4$. Again, SCC achieves zero error (see Table 4).

4.2.2 Face Clustering under Varying Lighting Conditions

We study the problem of clustering a given collection of images of human faces in fixed pose under varying illumination conditions. It has been proved that the set of all images of a Lambertian object under a variety of lighting conditions form a convex polyhedral cone in the image space, and this cone can be accurately approximated by a low-dimensional linear subspace (of dimension at most 9) (Ho et al. 2003; Basri and Jacobs 2003; Epstein et al. 1995). If we assume that images of different faces lie in different subspaces, then we can cluster these images by segmenting an arrangement of linear subspaces using SCC (and also LSCC).

Following Vidal et al. (2005) we use a subset of the Yale Face Database B (Georghiades et al. 2001) consisting of the frontal face images of three subjects (numbered by 5, 8, and 10) of the ten (see Fig. 5) under 64 varying lighting conditions. There are $N = 64 \times 3$ images in total. For computational efficiency, we have downsampled each image to 120×160 pixels, so the dimension of the image space is $D' = 120 \times 160$. We then stack these images (after vectorized) into a $D' \times N$ matrix $\mathbf{X}$ and apply SVD to reduce

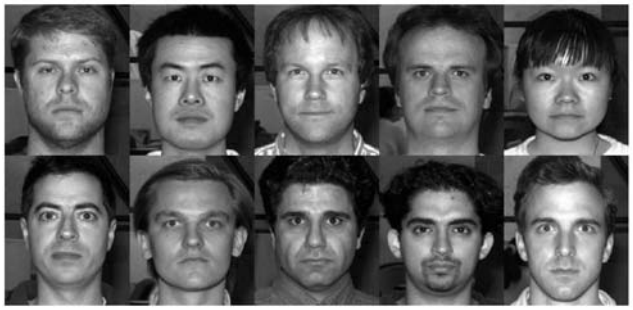


Fig. 5 The ten subjects in the Yale Face Database B. *First row*: subjects 1 through 5; *second row*: subjects 6 to 10

Table 5 Combinations (d, D) with which SCC and GPCA achieves a perfect segmentation respectively. Here d is the dimension of the subspaces, while D is the ambient dimension

Methods	(d, D)
SVD+SCC	$(0, 2/3/4), (1, 3/4), 2 \leq d < D \leq 10$
SVD+LSCC	$(1, 3/4/5/7/8), 2 \leq d < D \leq 10$
SVD+GPCA	$(3, 5), (4, 6), (4, 7), (5, 7), (4, 8), (6, 8)$

the ambient dimension to $D \ll D'$, forming a new matrix $\mathbf{Y} \in \mathbb{R}^{D \times N}$.

We apply SCC to the columns of $\mathbf{Y}$ and cluster three d-dimensional linear subspaces in $\mathbb{R}^D$. The above theory indicates that d should be at most 9. We have tried all the possible combinations $0 \leq d < D \leq 10$. The pairs (d, D) with which SCC and LSCC give a perfect segmentation are listed in Table 5. In comparison, we have also applied the GPCA-voting algorithm to the columns of $\mathbf{Y}$ with $0 \leq d < D \leq 10$. There are many situations where GPCA does not give 100% accuracy but SCC does (see Table 5).

Vidal et al. (2005) suggest to first project the data onto the top three principal components and then apply GPCA to the data in homogeneous coordinates by fitting three linear subspaces of dimensions 3, 2, and 2 in $\mathbb{R}^4$. They obtain zero error in this case. However, we are not aware of the reason of using mixed dimensions. We follow their strategy but instead we apply GPCA using the same dimension 3 for each linear subspace. Then a segmentation error of about 4% is obtained. We note that applying GPCA with $d = 3$ for each linear subspace (in homogeneous coordinates) in $\mathbb{R}^4$ is equivalent to applying SCC with $D = 3$ and $d = 2$. In this case, SCC achieves a perfect segmentation.

4.2.3 Temporal Segmentation of Video Sequences

We consider the problem of partitioning a long video sequence into multiple short segments corresponding to different scenes. We assume that all the image frames having the same scene live in a low dimensional subspace of the image space, and that different scenes correspond to different

Fig. 6 The first, 56th and last (135th) frames of the Fox video sequence

Table 6 The pairs (d, D) with which each algorithm obtains 100% accuracy. Here D is the ambient dimension, while d is the dimension of the subspaces

Method	(d, D)
SVD+SCC	$(0, 1/2/3/4), (1, 3/4), (2, 3/4/5)$
SVD+LSCC	$(1, 3), (2, 3/4), (3, 4)$
SVD+GPCA	NONE

subspaces. We show that the SCC and LSCC algorithms can be applied to solve this problem.

We borrow the video sequence from Vidal et al. (2005), which is about an interview at Fox TV (Fig. 6). It consists of 135 images of size 294×413, each containing either the interviewer alone, or the interviewee alone, or both. We would like to segment these images into the three scenes. We view each image frame as a sample point in $\mathbb{R}^{D'}$, where $D' = 294 \times 413$. We first apply SVD to reduce the ambient dimension from D' to $D \leq 10$, and then apply SCC/LSCC to segment three d-dimensional linear subspaces within $\mathbb{R}^D$. The combinations (d, D) with which SCC/LSCC obtains 100% accuracy are reported in Table 6.

Vidal et al. (2005) applied GPCA to solve this problem and obtained 100% accuracy. We do not know what dimensions of the ambient space and the subspaces they used. We also apply GPCA to segment d-dimensional linear subspaces in the projected space $\mathbb{R}^D$, where $1 \leq d < D \leq 10$. However, we did not find any combination that leads to a perfect segmentation.

5 Conclusions and Future Work

We have formed the Spectral Curvature Clustering (SCC) algorithm by introducing various techniques and combining them with its preliminary theoretical version (Chen and Lerman 2007). The complexity of the algorithm, i.e., the storage and running time, depends linearly on both the size of the data and the ambient dimension. We have performed extensive simulations to compare our algorithm with a few other standard methods. It seems that our algorithm is at least comparable to its competitors. It has a marked advantage in the case of affine subspaces and in certain instances of mixed dimensions. We have also applied our algorithm to several real-world problems, and obtained satisfactory results in all cases. Our algorithm performs well even in relatively high dimensional projected spaces, sometimes including the full space, and thus does not require aggressive dimensionality reduction as other algorithms.

Our work suggests many interesting future directions. In addition to those mentioned in Chen and Lerman (2007), this paper suggests the following avenues:

- **Justification of Iterative Sampling:** Our heuristic idea of iterative sampling seems to work well in all cases and thus results in a fast and accurate algorithm. We are interested in a more rigorous foundation for this procedure, in particular, finding conditions under which it converges (e.g., how large c should be to ensure convergence).
- **Improving the Outliers Detection Strategy:** We are interested in improving the strategy for detecting outliers, especially when the outliers percentage is not given. We would also like to study the robustness of the SCC algorithm.
- **Improving the Case of Mixed Dimensions:** Currently, when dealing with mixed dimensions, we use the highest dimension. This strategy works well in terms of $e_{\%}$. To improve the performance of SCC in this case, and consequently to more accurately evaluate the other error e_{OLS}, we plan to explore estimation of the true dimensions of the detected flats. Another strategy might be to hierarchically perform SCC according to different intrinsic dimensions.
- **Determining the Number of Flats and Their Dimensions:** Throughout this paper we have assumed that K and d_k are given. In many cases prior knowledge of these parameters may not be available. We thus need to develop techniques and criterions to select an optimal model.

Acknowledgements We thank the anonymous referees and the action editor for their valuable comments that have helped improve this paper, Tyler Whitehouse for commenting on an earlier version of this paper, Guillermo Sapiro for providing valuable references, Sameer Agarwal, Amnon Shashua, and Ron Zass for providing us their codes and for helpful email correspondence, Fei Yang for the contribution to the application of SCC to motion segmentation, Rene Vidal and Rizwan Chaudry for sending us the FOX video data. Special thanks go to Stefan Atev for insightful discussions and great help with programming. We are also grateful to IPAM and Mark Green for inviting us to participate in various programs on extracting information from high dimensional data and multiscale geometry. Those programs had a strong effect on the research described here. GL thanks Ingrid Daubechies who encouraged him to think about hybrid linear modeling already in 2002 (motivated by a brain imaging problem). This work was supported by NSF grant #0612608.

References

Agarwal, S., Branson, K., & Belongie, S. (2006). Higher order learning with graphs. In *Proceedings of the 23rd international conference on machine learning* (Vol. 148, pp. 17–24).

Agarwal, S., Lim, J., Zelnik-Manor, L., Perona, P., Kriegman, D., & Belongie, S. (2005). Beyond pairwise clustering. In *Proceedings of the 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)* (Vol. 2, pp. 838–845).

Bader, B., & Kolda, T. (2004). *Matlab tensor classes for fast algorithm prototyping* (Technical Report SAND2004-5187). Sandia National Laboratories.

Basri, R., & Jacobs, D. (2003). Lambertian reflectance and linear subspaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *25*(2), 218–233.

Bradley, P., & Mangasarian, O. (2000). k-plane clustering. *Journal of Global Optimization*, *16*(1), 23–32.

Brand, M. (2003). Fast online SVD revisions for lightweight recommender systems. In *Proc. SIAM international conference on data mining*.

Chen, G., & Lerman, G. (2007, submitted). Curvature-driven diffusion and hybrid flat-surfaces modeling. *Foundations of Computational Mathematics*. Latest version available at the supplementary webpage.

Drineas, P., Kannan, R., & Mahoney, M. (2006). Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication. *SIAM Journal on Computing*, *36*(1), 132–157.

Epstein, R., Hallinan, P., & Yuille, A. (1995). 5 ± 2 eigenimages suffice: An empirical investigation of low-dimensional lighting models. In *IEEE workshop on physics-based modeling in computer vision* (pp. 108–116).

Georghiades, A., Belhumeur, P., & Kriegman, D. (2001). From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *23*(6), 643–660.

Govindu, V. (2005). A tensor decomposition for geometric grouping and segmentation. In *Proceedings of the 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)* (Vol. 1, pp. 1150–1157).

Gruber, P., & Theis, F. (2006). Grassmann clustering. In *Proc. EUSIPCO 2006*. Florence, Italy.

Haro, G., Randall, G., & Sapiro, G. (2006). Stratification learning: Detecting mixed density and dimensionality in high dimensional point clouds. *Neural Information Processing Systems*.

Ho, J., Yang, M., Lim, J., Lee, K., & Kriegman, D. (2003). Clustering appearances of objects under varying illumination conditions. In *Proceedings of international conference on computer vision and pattern recognition* (Vol. 1, pp. 11–18).

Hyvärinen, A., & Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural Networks*, *13*(4–5), 411–430.

Kanatani, K. (2001). Motion segmentation by subspace separation and model selection. In *Proc. of 8th ICCV* (Vol. 3, pp. 586–591). Vancouver, Canada.

Kanatani, K. (2002). Evaluation and selection of models for motion segmentation. In *7th ECCV* (Vol. 3, pp. 335–349).

Lerman, G., & Whitehouse, J. T. (2008a). On d-dimensional d-semimetrics and simplex-type inequalities for high-dimensional sine functions. *Journal of Approximation Theory*. doi:10.1016/j.jat.2008.03.005. See also http://front.math.ucdavis.edu/0805.1430.

Lerman, G., & Whitehouse, J. T. (2008b). High-dimensional Menger-type curvatures—part I: Geometric multipoles and multiscale inequalities. http://front.math.ucdavis.edu/0805.1425.

Lerman, G., & Whitehouse, J. T. (2008c). High-dimensional Menger-type curvatures—part II: d-Separation and a menagerie of curvatures. http://front.math.ucdavis.edu/0809.0137.

Lerman, G., & Whitehouse, J. T. (2008d, in preparation). Least squares approximations for probability measures via multi-way curvatures. Will appear at the supplementary webpage once ready.

Ma, Y., Yang, A. Y., Derksen, H., & Fossum, R. (2008). Estimation of subspace arrangements with applications in modeling and segmenting mixed data. *SIAM Review*, *50*(3), 413–458.

Medioni, G., Lee, M.-S., & Tang, C.-K. (2000). *A computational framework for segmentation and grouping*. Amsterdam: Elsevier.

Ng, A., Jordan, M., & Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems* (Vol. 14).

Shashua, A., Zass, R., & Hazan, T. (2006). Multi-way clustering using super-symmetric non-negative tensor factorization. In *ECCV06* (Vol. IV, pp. 595–608).

Souvenir, R., & Pless, R. (2005). Manifold clustering. In *The 10th international conference on computer vision (ICCV 2005)*.

Sugaya, Y., & Kanatani, K. (2004). Multi-stage unsupervised learning for multi-body motion segmentation. *IEICE Transactions on Information and Systems*, *E87–D*(7), 1935–1942.

Tipping, M., & Bishop, C. (1999). Mixtures of probabilistic principal component analysers. *Neural Computation*, *11*(2), 443–482.

Tseng, P. (1999). *Nearest q-flat to m points* (Technical report).

Vidal, R., Ma, Y., & Sastry, S. (2005). Generalized principal component analysis (GPCA). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(12), 1945–1959.