Incremental Kernel PCA for Efficient Non-linear Feature Extraction

Tat-Jun Chin David Suter
Institute for Vision Systems Engineering
Monash University, Victoria, Australia.
{tat.chin | d.suter}@eng.monash.edu.au

Abstract

The Kernel Principal Component Analysis (KPCA) has been effectively applied as an unsupervised non-linear feature extractor in many machine learning applications. However, with a time complexity of $O(n^3)$, the practicality of KPCA on large datasets is minimal. In this paper, we propose an approximate incremental KPCA algorithm which allows efficient processing of large datasets. We extend a linear PCA updating algorithm to the non-linear case by utilizing the kernel trick, and apply a reduced set construction method to compress expressions for the derived KPCA basis at each update. In addition, we show how multiple feature space vectors can be compressed efficiently, and how approximated KPCA bases can be re-orthogonalized using the kernel trick. The proposed method is justified through experimental validations.

1 Introduction

The KPCA method [9] can be applied as an unsupervised non-linear feature extractor in domains where linear PCA is effective but has somewhat reached a performance barrier due to linear assumptions of the underlying generative phenomena. KPCA allows estimation of non-linear kernel principal components (KPCs) which are more suitable to describe highly complex and non-linear data distributions such as face images, handwritten digits and natural images. In applications where comparisons have been done, KPCA almost always outperforms PCA.

However, in order to obtain accurate non-linear principal components from complex data distributions, large training datasets are required, especially for data embedded in a high-dimensional space. This presents a difficulty for KPCA since it has to store and manipulate all data at once. Secondly, the resulting KPCs have to be defined implicitly by linear expansions of the training data, thus all data must be saved after training. For massive datasets, this means high costs for storage resources and computational load during utilization of KPCs. Furthermore, for applications that require online data processing, KPCA is impracticable since it is computable in a batch manner only. The problem of batch KPCA training on large datasets is contingent upon the storage and processing of the kernel matrix M. Given an n-vector training dataset, the size of M is n^2. Secondly, invoking an eigendecomposition on M to perform KPCA involves a time complexity of $O(n^3)$. This can severely handicap KPCA training on large datasets.

To solve this problem, we propose an approximate incremental KPCA algorithm. Our idea involves kernelizing an *exact* linear PCA updating algorithm and applying a reduced set (RS) construction method [9] at each iteration so as to maintain constant processing speed and memory usage. We show how to efficiently compress multiple feature space vectors via RS construction, and how approximated KPCA bases can be re-orthogonalized. Experimental results demonstrate the effectiveness of the method.

2 Related Work

The work most similar in *objective* to this paper is [5], where an approximate incremental KPCA solution was proposed. The method involves mapping the training data via the Empirical Kernel Map [9] with the intended kernel and performing a PCA updating algorithm on the mapping results. A major drawback of their approach is that novel data *cannot* be incorporated. Hence, it is incremental only in the sense that not all data have to be considered at once, but *all* data must be available initially to define the Empirical Kernel Map (this means the method *cannot* be applied for online computations). Additionally, a few methods (e.g. Nyström method, Greedy KPCA [9]) have been proposed to tackle the computational complexity problem of KPCA. Our solution differs from these in that not only it reduces computational time complexity, it has the capability of *updating* with novel data unavailable initially in a way that maintains non-increasing memory usage and update duration. This is crucial for real-time applications.

In [6], the Kernel Hebbian Algorithm (KHA) was proposed as an iterative KPCA algorithm. Essentially, the method involves kernalizing the Generalized Hebbian Algorithm (GHA) which is an online computation procedure for linear PCA. Similar in operational characteristics to single-layer feedforward neural networks, the KHA outputs converge towards KPCs by *iterating* the algorithm using the intended training data over *multiple passes*. Our work differs in that we seek an *incremental* computation algorithm for KPCA such that it is unnecessary to consider all available data more than once.

In [2], an incremental kernel SVD (KSVD) algorithm was proposed which involves kernelizing incremental linear SVD. This does not imply incremental KPCA is solved, since incremental KSVD does not require adaptive centering of temporal data and its effect on KPCs. Here, our work kernelizes a *different* underlying linear algorithm. Note that KPCA and KSVD return vastly different results on the same dataset. Secondly, their RS compression scheme is poor causing severe subspace drift. Thirdly, their approximated KSVD basis is not re-orthogonalized, exacerbating drift. Here, we demonstrate a superior compression strategy and a kernel subspace re-orthogonalization scheme.

In [8], KPCA was used to obtain feature descriptors from multiple images for application in mobile robot navigation and localization. RS expansions are constructed to compress the KPCA-derived bases to reduce computational load during KPC utilization. However, no method was proposed to incrementally update the principal components as more features become available. Hence, their's is not an incremental KPCA solution.

3 The Kernel PCA

We first establish the meaning of commonly used symbols. Given a matrix M, the symbol $M_{a:b,c:d}$ defines the submatrix that contains the elements of M within the intersection of the a-th row till the b-row and the c-th column till the d-th column. If the row or column specifiers are omitted, e.g. $M_{:,c:d}$, take all available rows or columns.

We begin by obtaining a data matrix $a = [x_1 \ \cdots \ x_n] \in \mathbb{R}^{m\times n}$, with $x_i \in \mathbb{R}^m$ being the i-th input data. For KPCA, we non-linearly map a to a higher dimensional space $\mathcal{F}$ using the function $\phi : \mathbb{R}^m \longrightarrow \mathcal{F}$ and perform PCA in $\mathcal{F}$. The map ϕ is induced by a kernel function $k(\cdot\,,\cdot)$ that evaluates inner products in $\mathcal{F}$:

$$k(x,z) = \phi(x)\cdot\phi(z)\ ,\ \text{ with } \ x,\, z \in \mathbb{R}^m\ . \tag{1}$$

If $k(\cdot\,,\cdot)$ is an appropriately chosen *Mercer* kernel, then ϕ belongs to a function space that has the structure of a so-called *Reproducing Kernel Hilbert Space* (RKHS). See [9, 3] for more details. Using ϕ, we transform a into $A = [\phi(x_1) \ \cdots \ \phi(x_n)]$. We center A by subtracting the mean μ_A from A. The mean and mean-adjusted data are respectively

$$\mu_A = A\left(\frac{1}{n}\mathbf{1}_{n,1}\right) := Av \ \text{ and } \ \hat{A} = A\left(\mathbf{I}_n - v\,\mathbf{1}_{1,n}\right) := Av'\ . \tag{2}$$

$\mathbf{I}_n$ indicates an identity matrix of size $n\times n$ while $\mathbf{1}_{r,c}$ represents a matrix of ones of size $r\times c$. Consider $M = \hat{A}^T\hat{A} = (v')^T A^T A v'$ and its eigenvalue decomposition $M = Q\Delta Q^T$. By using the kernel function, $A^T A$ can be evaluated without having to perform the mapping ϕ. Via *kernel singular value decomposition* (KSVD) [3], the rank-r singular value factorization of $\hat{A}$ is

$$\hat{A}^r = \left[\hat{A}Q^r(\Delta^r)^{-\frac{1}{2}}\right]\left[(\Delta^r)^{\frac{1}{2}}\right]\left[(Q^r)^T\right] \equiv U^r\Sigma^r(V^r)^T\ , \tag{3}$$

where $Q^r = Q_{:,1:r}$ and $\Delta^r = \Delta_{1:r,1:r}$. Refer to [3] for proofs. $A^T A$ (and hence M) is positive semi-definite if it is constructed using a Mercer kernel. The columns of U^r are the first-r PCs of A (i.e. the KPCs of a) ranked according to $\sqrt{\Delta^r}$. Observe that the PCs are defined implicitly by linear expansions of mapped input data:

$$U^r = Av'Q^r(\Delta^r)^{-\frac{1}{2}} := A\alpha\ . \tag{4}$$

Given a point $\phi(z)$ in $\mathcal{F}$ corresponding to mapped input data z, we center and project it onto U^r via

$$(U^r)^T(\phi(z)-\mu_A) = (\alpha)^T A^T \left[\ A \quad \phi(z)\ \right]\begin{bmatrix} -v \\ 1 \end{bmatrix}\ , \tag{5}$$

where $A^T[\,A \ \ \phi(z)\,]$ can be evaluated using the kernel function since it contains dot products between feature space vectors. The projection components are considered extracted non-linear features. For an alternative but equivalent interpretation of the process of carrying out KPCA, refer to [9].

4 Incremental Kernel PCA

In this section we propose an approximate KPCA updating scheme. An ingenious method for updating linear PCA with incremental data was proposed by [7]. Essentially it utilizes an incremental SVD computation procedure coupled with a shift of the overall sample mean in consideration of new data to update a previous basis produced via PCA. We extend their algorithm to enable KPCA updating. Note that although the following derivations make references to feature space vectors, their explicit existence is never required and the mapping ϕ is always avoided.

4.1 Updating a Previous Factorization

To begin, assume we have data $a \in \mathbb{R}^{m\times n}$ with r KPCs U^r and mean μ_A. These are defined as linear expansions of mapped input data (refer to Section 3):

$$U^r = A\alpha\ ,\ \ \mu_A = A\nu\ , \tag{6}$$

with $A = \phi(a)$. Assume that the corresponding singular values Σ^r and right singular vectors V^r obtained during KSVD are available as well. We define $\dot{A}^r$ as the reconstruction of A using the first-r PCs i.e. $\dot{A}^r = \hat{A}^r + \mu_A$.

Given new data $b \in \mathbb{R}^{m\times c}$, the PCA of the overall data $D = [\,\dot{A}^r\ B\,]$ is sought, where $B = \phi(b)$. The mean of B is

$$\mu_B = B\omega\ ,\ \text{ with }\ \omega = \frac{1}{c}\mathbf{1}_{c,1}\ . \tag{7}$$

The mean of the overall data D can be updated as

$$\mu_D = \frac{n}{n+c}A\nu + \frac{c}{n+c}B\omega := \bar{A}\bar{\nu}\ ,\ \text{ with }\ \bar{A} = [\,A\ B\,]\ \text{ and }\ \bar{\nu} = \frac{1}{n+c}\begin{bmatrix} n\nu \\ c\omega \end{bmatrix}\ . \tag{8}$$

Following Equation (2), B is centered with regards to its own mean μ_B as

$$\hat{B} = B\left(\mathbf{I}_c - \omega\,\mathbf{1}_{1,c}\right) := B\omega'\ . \tag{9}$$

At this stage, we use these intermediate results to construct matrix $\tilde{E}$ which is defined as

$$\tilde{E} = \left[\ \hat{B}\quad \sqrt{\tfrac{nc}{n+c}}(\mu_A - \mu_B)\ \right] = [\,A\ B\,]\left[\ \left[\begin{smallmatrix}\mathbf{0}_{n,c} \\ \omega'\end{smallmatrix}\right]\quad \sqrt{\tfrac{nc}{n+c}}\left[\begin{smallmatrix}\nu \\ -\omega\end{smallmatrix}\right]\ \right] := \bar{A}\gamma\,, \tag{10}$$

with $\mathbf{0}_{n,c}$ indicating an $n\times c$ matrix of zeroes. We can see that μ_D and $\tilde{E}$ are linearly expanded from mapped input data A and B.

To perform PCA on D, we can center D with regards to its own mean μ_D and invoke the SVD. However, this would correspond to a batch PCA since both $\dot{A}^r$ and B have to be utilized. For incremental PCA computation, we would not want to access $\dot{A}^r$. To this end, it was algebraically proven in [7] that the scatter matrix corresponding to D is

$$S_D = \hat{D}\hat{D}^T = [\,\hat{A}^r\ \ \tilde{E}\,][\,\hat{A}^r\ \ \tilde{E}\,]^T\ , \tag{11}$$

where $\hat{D} = D - \mu_D$. In other words, to perform PCA on D, it is sufficient to carry out an SVD on $[\,\hat{A}^r\ \ \tilde{E}\,]$. At this stage, we apply incremental SVD procedures to update U^r using $\tilde{E}$ to avoid accessing $\hat{A}^r$. Given the previous factorization $\hat{A}^r = U^r\Sigma^r(V^r)^T$, we decompose $[\,\hat{A}^r\ \ \tilde{E}\,]$ as

$$\left[\ \hat{A}^r\quad \tilde{E}\ \right] = \left[\ U^r\quad J\ \right]\begin{bmatrix} \Sigma^r & L \\ \mathbf{0}_{c+1,r} & K \end{bmatrix}\begin{bmatrix} V^r & \mathbf{0}_{n,c+1} \\ \mathbf{0}_{c+1,r} & \mathbf{I}_{c+1} \end{bmatrix}^T\ , \tag{12}$$

with $L = (U^r)^T\tilde{E}$, $H = \tilde{E} - U^rL$ and $JK \xleftarrow{\text{QR}} H$. Let E, F and G^T be respectively defined as the left, middle and right matrix of the right-hand-side of (12). We can diagonalize F by invoking the SVD, i.e. $F = U'\Sigma'\left(V'\right)^T$, and substitute it into (12), yielding the updated SVD:

$$\left[\ \hat{A}^r\quad \tilde{E}\ \right] = U''\Sigma'(V'')^T\ , \tag{13}$$

with $U'' = EU'$ and $V'' = GV'$. The first r PCs can be updated via $U^r \longleftarrow U''_{:,1:r}$. Note that this is achieved without involving $\hat{A}^r$ or $\dot{A}^r$ and is dependent only on $\tilde{E}$, U^r and Σ^r. V^r can be discarded for applications that do not make use of it. See [1] for details of incremental SVD.

4.2 Applying the Kernel Trick

The task now is to compute J, K and L without having to access feature space vectors directly. Matrix L is defined as

$$L = (U^r)^T \tilde{E} = \alpha^T A^T \bar{A} \gamma \tag{14}$$

with $L \in \mathbb{R}^{r\times(c+1)}$. L is computable by using the kernel function for inner products between the columns of A and $\bar{A}$. We can define matrix H subsequently as:

$$H = \tilde{E} - U^r L = [\, A\ B \,] \begin{bmatrix} \gamma_{1:n,:} - \alpha L \\ \gamma_{(n+1):(n+c),:} \end{bmatrix} := \bar{A}\beta \ . \tag{15}$$

Again, H is expressible as linear combinations of mapped input data with $\beta \in \mathbb{R}^{(n+c)\times(c+1)}$ containing the expansion coefficients. Instead of using the QR decomposition to obtain an orthonormal basis J for H, we derive an equivalent orthonormal basis by performing a KSVD on H and retain all left singular vectors to form J. We can compute the kernel matrix for H as

$$M_H = \beta^T \bar{A}^T \bar{A} \beta = \beta^T \bar{M} \beta \ , \tag{16}$$

with $M_H \in \mathbb{R}^{(c+1)\times(c+1)}$. $\bar{M}$ can be evaluated using the kernel function on input data from A and B. Hence, $\bar{M}$ will be positive semi-definite and this will ensure the positive semi-definiteness of M_H as well. In addition, the rank of M_H is equal to the rank of H. Let the eigenvalue decomposition of $M_H = Q_H \Delta_H Q_H^T$. From Equation (3), the matrix J and K would then be

$$J = \bar{A}\beta\, Q_H \Delta_H^{-\frac{1}{2}} := \bar{A}\Omega \ , \quad K = \Delta_H^{\frac{1}{2}} Q_H^T \ , \tag{17}$$

with $\Omega \in \mathbb{R}^{(n+c)\times(c+1)}$ and $K \in \mathbb{R}^{(c+1)\times(c+1)}$. Note that H can be rank deficient due to the lack of novel information in B. In fact, H is always rank deficient by at least 1 due to data centering of B. Thus, the eigenvectors with zero eigenvalue should be ignored since they do not contribute towards describing $\mathrm{span}(H)$. To do this, retain only the first $\mathrm{rank}(M_H)$ columns of Ω and $\mathrm{rank}(M_H)$ rows of K. Matrix F is then constructed as shown in Equation (12) and diagonalized to result in $F = U'\Sigma'(V')^T$. The left singular vectors of the matrix $[\, \hat{A}^r\ \tilde{E} \,]$ are

$$U'' = \begin{bmatrix} A\alpha & \bar{A}\Omega \end{bmatrix} U' := \bar{A}\Psi \ , \tag{18}$$

$$\text{with} \quad \Psi = \begin{bmatrix} \alpha \\ \mathbf{0}_{c,r} \end{bmatrix} U'_{1:r,:} + \Omega U'_{(r+1):(r+\mathrm{rank}(M_H)),:} \ .$$

The first r KPCs of a are then revised as $U^r \longleftarrow \bar{A}\bar{\alpha}$, where $\bar{\alpha} = \Psi_{:,1:r}$. Along with the overall data mean μ_D, this is the updated KPCA.

5 Compressing Feature Space Vector Expansions

Up to this stage our incremental KPCA computation is still exact, but observe that in this procedure it is unavoidable that the updated KPCA basis and data mean are expanded from old and new mapped image vectors. Thus, although we are spared from computing a batch KPCA at every iteration, we still have to store all seen data. This is detrimental towards maintaining constant update speed and memory usage. To solve this problem, we compress the feature vector representations by constructing reduced set (RS) expansions. To this end, various approaches (see [9]) can be applied, but here we use the “iterated pre-images” method. The following provides a generic description of the underlying idea.

5.1 Constructing RS Expansions

Suppose we have a feature space vector $u \in \mathcal{F}$ (e.g. KPCs, data mean) linearly expanded from n mapped input data $\{x_1, \cdots, x_n\} \in \mathbb{R}^m$:

$$u = \sum_{i=1}^{n} \alpha_i \, \phi(x_i) \; . \tag{19}$$

Here, $\phi : \mathbb{R}^m \to \mathcal{F}$ denotes the mapping induced by a kernel function and α_i are the expansion coefficients. We seek a *pre-image* $y \in \mathbb{R}^m$ so that $u = \phi(y)$. Most likely an exact pre-image will not exist (see [9]), and we will have to approximate u with a series of approximate pre-images:

$$u \approx \sum_{i=1}^{n'} \alpha'_i \, \phi(y_i) \; , \tag{20}$$

where $n' < n$ and α'_i are the new expansion coefficients. The intricacies on how to construct this RS expansion are beyond the scope of this paper, but note that the larger n' is the smaller the discrepancy of the approximation. Refer to [9] for details.

5.2 Compressing Several Feature Space Vectors Simultaneously

At each iteration, we have $r+1$ feature space vectors (r KPCs and one data mean) expanded from a library of n mapped input data. By building RS expansions, we compress the representation of the feature space vectors. Note that RS expansions need to be constructed for each vector individually. The current library of n expansion vectors is overwritten with $(r+1)n'$ pre-images which will be used for future updates. Naturally, compression is worthwhile only if $n > (r+1)n'$. For each feature space vector, coefficients for unrelated pre-images can be set to zero. See Figure 1(a).

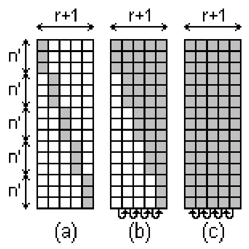


Figure 1: Each block represents the matrix of linear expansion coefficients of a set of pre-images estimated for RS expansions of $r+1$ feature space vectors. Each square within a block indicates one coefficient, with grey squares representing non-zero coefficients and white otherwise. (a) Pre-images are inefficiently used. (b) Pre-images of a vector are used to aid in spanning a subsequent vector. (c) Second pass allows full use of pre-images.

The previous approach tends to be wasteful, since the mapped pre-images in the approximation of one feature space vector will unlikely be orthogonal to the other vectors in feature space $\mathcal{F}$, and can be used essentially for free to aid in approximating the other vectors [9]. This means that if u_2 is compressed after u_1, these feature space vectors will have the form

$$u_1 \approx \sum_{i=1}^{n'} \alpha_i' \phi(y_i) \ , \ \ u_2 \approx \sum_{i=1}^{n'} \beta_i' \phi(y_i) + \sum_{i=1}^{n'} \gamma_i' \phi(z_i) \ , \tag{21}$$

where β_i' are determined for y_i with respect to approximating u_2, while new pre-images z_i with coefficients γ_i' are estimated to further improve the approximation accuracy. The pre-image set (y_i, z_i) is then used to assist in compressing u_3. Figure 1(b) illustrates this.

This paper extends this concept further to incorporate a second pass of coefficient estimation. After the first stage of sequential compression, an extra boost of approximation accuracy is achievable if we re-estimate the coefficients for each feature space vector expansion with regards to the overall pre-image set constructed during the first stage of compression. Although without theoretical guarantee, in practice, an improvement over the first pass is almost always obtained. This is despite most of the pre-images not being specifically tailored to span a particular feature space vector. See Figure 1(c).

5.3 Re-Orthogonalizing the KPCA Basis

Most likely a RS-expanded basis $\tilde{U}^r = A'\alpha'$ of an original KPCA basis $U^r = A\alpha$ is non-orthogonal due to approximation errors. To re-orthogonalize $\tilde{U}^r$ we can use KSVD again, with

$$M_o = {\alpha'}^T {A'}^T A'\alpha' = Q_o D_o Q_o^T \tag{22}$$

as the kernel matrix on which we invoke an eigenvector decomposition. The orthogonalized basis is then $A'\alpha' Q_o D_o^{-\frac{1}{2}}$ on which we project the approximated basis via

$$P = (\alpha' Q_o D_o^{-\frac{1}{2}})^T (A')^T A'\alpha' \ . \tag{23}$$

Thus, the approximated basis is expanded using the orthogonal basis as

$$\tilde{U}_o^r = A'\alpha' Q_o D_o^{-\frac{1}{2}} P := A'\pi \ . \tag{24}$$

To normalize the basis vectors, the columns of P are normalized. To express the singular values corresponding to the new basis, project the original singular values onto this basis:

$$S = (\tilde{U}_o^r)^T U^r \Sigma^r = \pi^T (A')^T A\alpha\Sigma^r \ . \tag{25}$$

The singular values associated with the new basis is $\tilde{\Sigma}^r = \mathrm{diag}(S)$. The non-zero off-diagonal elements of S discarded due to the $\mathrm{diag}(\cdot)$ operation correspond to the approximation errors of the RS approximation. Note that the process does not require explicit evaluations of ϕ since only dot products between feature space vectors are required.

6 Experimental Results

We define IKPCA as an acronym of the proposed incremental KPCA algorithm with RS constructions. 2D synthetic problems were considered first. The problems were generated in the following way (see Figure 2): x-values have uniform distribution in $[-1, 1]$, y-values are generated from $y = x^2 + \xi$, where ξ is normal noise with standard deviation 0.2. From Figure 2, no visually discernible differences can be observed in the results of batch KPCA and IKPCA where the RBF kernel with $\sigma = 1$ was used. Table 1 depicts training durations of KPCA, IKPCA and KHA[1] (using 2nd degree polynomial kernel) on synthetic problems of various sizes. As expected, batch KPCA scales with complexity $O(n^3)$. In contrast, IKPCA scales almost linearly, while KHA appears to scale badly. All programs were run on Matlab on a 2.8Ghz Pentium 4 machine with 512MB of memory.

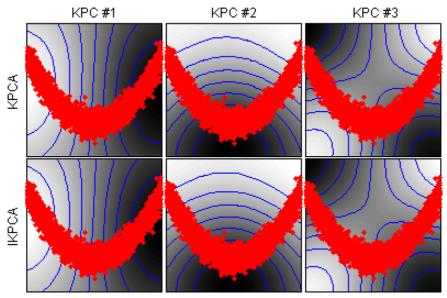


Figure 2: A 3100-vector 2D synthetic problem. Contour lines show constant values of projection onto KPCs.

Vectors	KPCA	IKPCA	KHA
520	7.23s	29.05s	93.75s
1000	44.10s	60.23s	198.73s
1510	161.23s	93.14s	381.92s
2020	413.05s	127.45s	695.31s
2530	990.29s	141.28s	990.86s
3100	1394.91s	165.29s	1569.02s

Table 1: Training durations on synthetic problems of various sizes. Note that what is compared here is the empirical time complexity, i.e. the rate of growth of training duration with increase of problem size.

Next, popular datasets[2] for manifold learning algorithms were considered. In particular, the perforated Swiss-roll (Figure 3(a)), rotating Teapot (Figure 3(b)) and the Frey Face video sequence (Figure 3(c)) were trained using IKPCA. The eigenspectrum of the datasets induced by the respective choice of kernels (see Figure 3(d)) show that none of the datasets (except Swiss-roll) occupy an inherently low-dimensional subspace of their corresponding feature space. Hence, during training the number of KPCs is selected to encompass as much variance as possible (to prevent severe KPC drift) while maintaining reasonable training speed. After training, each dataset is projected onto the first-10 KPCs. The same process is repeated for batch KPCA. Figures 4(a), 4(b) and 4(c) depict the results which show that no critical differences are observable from the projection components of the KPCs trained separately using batch KPCA and IKPCA.

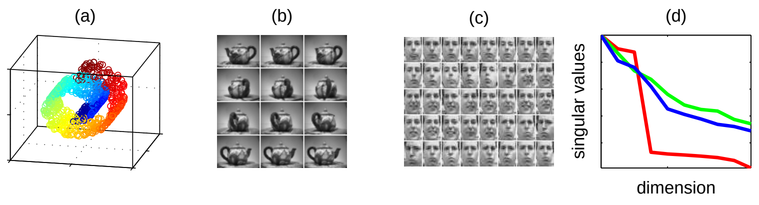
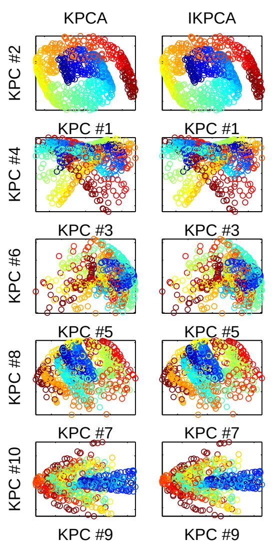
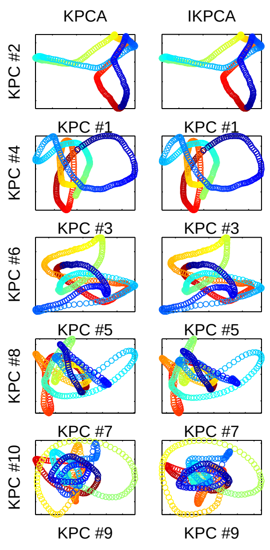


Figure 3: (a) The 800-vector perforated Swiss-roll dataset. (b) Several samples of the 400-image Teapot dataset. (c) Several samples of the Frey Face dataset which contains 1965 sequential frames of a face from a video. (d) Eigenspectrum of the datasets. Red: Swiss-roll (RBF kernel with $\sigma = 1$), Green: Teapot (2nd degree polynomial kernel), Blue: Frey Face (2nd degree polynomial kernel).

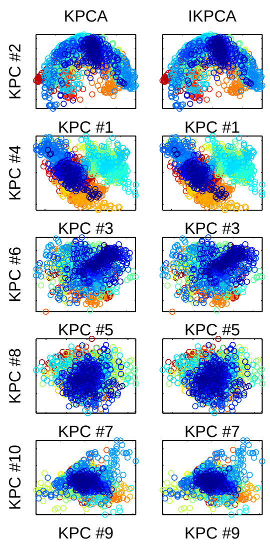
[1]Code obtained from http://www.kyb.tuebingen.mpg.de/~mof. The RBF kernel is not supported by the code.
[2]http://www.seas.upenn.edu/~kilianw/sde/datasets.htm



(a) Swiss-roll



(b) Teapot



(c) Frey Face

Figure 4: The first-10 KPC projection components for the manifold learning datasets. For each subfigure, the left column is the KPCA result, while the right is IKPCA. Each point is assigned a colour to facilitate visual comparison, and smooth colour changes represent continuous progression on the manifold. The figures are best viewed in colour.

As a practical application, we use IKPCA in the Kernel Eigenfaces [10] method for face recognition. The Yale Face Database B [4] which contains 5760 images of 10 subjects was used in the experiments. Each subject has 9 poses, and each pose is captured under 64 distinct lighting conditions. See Figure 5. The images were cropped and resized to 20x20 pixels. The dataset was then randomly partitioned into two disjoint subsets: Set 1 contained 5000 training images, and Set 2 contained 760 test images. By training Set 1 with IKPCA, we avoided storing a massive 5000x5000 kernel matrix for KPCA. By using $r = 36$, $n = 10$ and $c = 30$, the intermediate matrices L (36x31), β (40x31), M_H (31x31), Ω (40x31) and K (31x31) of IKPCA (see Section 4.2) are relatively tiny. For comparisons, a subset of Set 1 with 1000 images was trained using KPCA and IKPCA. The 2nd order polynomial kernel and the RBF kernel with $\sigma = 1$ were used since they gave the best results in [10]. For classification, the training and test images were projected onto the first-36 Kernel Eigenfaces (KPCs) and the k-nearest neighbour rule with $k = 10$ was evaluated. The results in Table 2 show that by being able to process more exemplars, the Kernel Eigenfaces obtained using IKPCA from Set 1 managed to outperform those estimated via batch KPCA on the 1000-vector subset of Set 1. Moreover, the performances of IKPCA and KPCA were almost identical given the same 1000-vector subset. Despite the modest accuracy compared to other face recognition methods, what is demonstrated here is the benefit of being able to process large datasets. In [10], Kernel Eigenfaces yielded a similar error rate of 27.27% $(45/165)$ on the much smaller Yale Face Database[3].

[3]http://cvc.yale.edu/projects/yalefaces/yalefaces.html

Figure 5: Face image samples.

Algorithm	Vectors	Kernel	Error rate (%)
KPCA	1000	rbf, $\sigma = 1$	65.92 (501/760)
IKPCA	1000	rbf, $\sigma = 1$	66.84 (508/760)
KPCA	1000	poly, $d = 2$	49.61 (377/760)
IKPCA	1000	poly, $d = 2$	50.26 (382/760)
IKPCA	5000	rbf, $\sigma = 1$	46.58 (354/760)
IKPCA	5000	poly, $d = 2$	26.97 (205/760)

Table 2: Kernel Eigenfaces classification results.

7 Conclusion

In this paper, we proposed an incremental algorithm for performing non-linear (kernel) PCA. This is achieved by kernelizing an exact linear PCA updating algorithm and using RS expansions to maintain constant update speed and memory usage. We showed how multiple KPCs can be accurately and efficiently compressed and how orthogonality of the approximated KPCA basis can be enforced. Through several experiments, we showed the accuracy and good scaling properties of the proposed algorithm. Finally, a face recognition task was demonstrated as practical application of the proposed method.

References

[1] M. Brand. Incremental singular value decomposition of uncertain data with missing value. In *ECCV*, pages 707–720, 2002.

[2] T.-J. Chin, K. Schindler, and D. Suter. Incremental kernel SVD for face recognition with image sets. In *IEEE FGR*, 2006.

[3] N. Cristianini and J. Shawe-Taylor. *Kernel methods for pattern analysis*. Cambridge University Press, 2004.

[4] A. S. Georghiades et al. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE PAMI*, 23(6):643–660, 2001.

[5] Byung-Joo Kim. Active visual learning and recognition using incremental kernel pca. In *Australian Conference on Artificial Intelligence*, pages 585–592, 2005.

[6] K. I. Kim, M. O. Franz, and B. Schölkopf. Iterative kernel principal component analysis for image modeling. *IEEE PAMI*, 27(9):1351–1366, 2005.

[7] J. Lim, D. Ross, R.-S. Lin, and M.-H. Yang. Incremental learning for visual tracking. In *Advances in NIPS*, pages 793–800, 2004.

[8] J. Meltzer, M.-H. Yang, R. Gupta, and S. Soatto. Multiple view feature descriptors from image sequences via kernel pca. In *ECCV*, pages 215–227, 2004.

[9] B. Schölkopf and A. Smola. *Learning with kernels*. The MIT press, 2002.

[10] M.-H. Yang. Kernel Eigenfaces vs. Kernel Fisherfaces: Face recognition using kernel methods. In *IEEE FGR*, pages 215–220, 2002.