Efficiently Updating and Tracking the Dominant Kernel Eigenspace

L. Hoegaerts[a], L. De Lathauwer[a,b], J.A.K. Suykens[a], J. Vandewalle[a]

[(a)] Katholieke Universiteit Leuven
Department of Electrical Engineering, ESAT-SCD-SISTA
Kasteelpark Arenberg 10, B-3001 Leuven (Heverlee), Belgium
Phone: +32-(0)16-32.85.40 - Fax: +32-(0)16-32.19.70
Email: {Luc.Hoegaerts, Johan.Suykens, Joos.Vandewalle}@esat.kuleuven.ac.be

[(b)] ETIS, UMR 8051 (CNRS, ENSEA, UCP)
Avenue du Ponceau 6, BP 44, F 95014 Cergy-Pontoise Cedex, France
Phone: +33-(0)1-30.73.66.10 - Fax: +33-(0)1-30.73.66.27
E-mail: Lieven.DeLathauwer@ensea.fr

Abstract

The dominant set of eigenvectors of the symmetric kernel Gram matrix is frequently used in many important kernel methods (like e.g. kernel Principal Component Analysis, feature approximation, denoising, compression, prediction) in the machine learning domain. Yet in the case of dynamic and/or large scale data, the batch calculation nature and computational demands of the eigenvector decomposition limit these methods in numerous applications. In this paper we present an efficient incremental approach for fast calculation of the dominant kernel eigenbasis, which allows to track the kernel eigenspace dynamically. Experiments show that our updating scheme delivers a numerically stable and accurate approximation at every iteration in comparison to the batch algorithm.

I. Introduction

Over the last years one can see many learning algorithms being transferred to a kernel representation [1], [2]. The benefit lies in the fact that nonlinearity can be allowed, while avoiding to solve a nonlinear optimization problem. The transfer is implicitly accomplished by means of a nonlinear map into an often high-dimensional Reproducing Kernel Hilbert Space H_k (RKHS) [3].

When n data points $\mathbf{x}_i \in \mathbb{R}^p$ are given, a square symmetric $n \times n$ kernel Gram matrix K with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ can be computed, where a kernel k provides a similarity measure between pairs of data points

$$k : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R} : (\mathbf{x}_i, \mathbf{x}_j) \mapsto k(\mathbf{x}_i, \mathbf{x}_j). \tag{1}$$

Many important methods (e.g. KPCA [4], FS-LSSVM [2], etc..) rely now, either directly or indirectly, on the dominant m eigenvectors of K, where $m \ll n$. The usual tool for computing this eigenspace is the singular value decomposition (SVD) because of its high numerical accuracy. But the drawback is that standard routines for the batch computation of an SVD require $O(n^3)$ operations (or $O(mn^2)$ for instance in deflation schemes).

It is clear that for large datasets (as for example in image processing, computer vision or object recognition), the kernel method with its powerful advantage of dealing with nonlinearity, is computationally severely limited. For large datasets an eigendecomposition of K can simply become too timeconsuming.

But also memory is an issue as $O(n^2)$ memory units are required. In addition, computations may have to be repeated several times because the problem is nonstationary. Some applications require adaptive algorithms. A truly online algorithm for extracting kernel principal components has not been reported so far, although the problem is pushing and attempts to tackle this problem are presently undertaken [5], [6], [7].

A literature that spreads out over the last three decades has spent considerable effort on dealing with these issues, with a common denominator called: SVD updating [8], [9], [10], [11], [12], [13], the process of using the previous eigenspace when a new data instance arrives. Most of them are methods in which the eigenspace is incrementally updated when more rows (or more columns) are added to the involved matrix.

Yet, the particular context of kernel algorithms, where the matrix dimensions correspond to the number of training data points, those updating schemes cannot be simply applied. The kernel matrix namely expands *both* in row and column dimension when a point is added. To our present knowledge, this case has not been addressed yet. In this paper we present a fast online updating and downdating procedure for the dominant eigenspace in case of a *growing* symmetric square matrix, with a computational complexity of $O(nm^2)$ and memory requirements of $O(nm)$ per iteration, and with very limited loss of accuracy. For the situation of subspace tracking, where one keeps the dimension of the subspace fixed, also an efficient downsizing mechanism is proposed.

In section II we propose an updating and downdating scheme for an existing eigenspace. In section III we develop an efficient way to downsize the row dimension of an updated eigenspace. In section IV we present experiments to show that accuracy and stability are preserved during the iteration.

II. Updating and downdating

In this section we describe the updating in detail for one step of the iterative procedure and show that it compares favorably to complexity of a batch SVD. Due to the particular structure of our updating scheme, downdating will just be a special case of updating.

A. Updating

For the updating we assume we are given the (approximate) dominant $n \times m$ eigenspace U_{nm} of the square symmetric $n \times n$ kernel matrix K_n, based on n data points. The goal of updating is to obtain the $(n+1) \times m$ eigenspace $U_{(n+1)m}$ of the $(n+1) \times (n+1)$ kernel matrix K_{n+1}, based on the previous n data points and one extra data point. In the kernel matrix an update is located in the last row and column, which expands K_n *both* in row and column dimension when a point is added:

$$K_{n+1} = \begin{bmatrix} K_n & \mathbf{a} \\ \mathbf{a}^T & b \end{bmatrix} \tag{2}$$

where $\mathbf{a}$ is a $n \times 1$ vector of kernel entries $a_i = k(\mathbf{x}_i, \mathbf{x}_{n+1})$ and scalar $b = k(\mathbf{x}_{n+1}, \mathbf{x}_{n+1})$.

The key observation here is that K_{n+1} can be expanded as a stripped matrix to which a difference of 2 rank-one matrices is added. We propose the following convenient expansion:

$$K_{n+1} = \begin{bmatrix} K_n & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} + \frac{1}{b} \begin{bmatrix} \mathbf{a} \\ b \end{bmatrix} \begin{bmatrix} \mathbf{a}^T & b \end{bmatrix} - \frac{1}{b} \begin{bmatrix} \mathbf{a} \\ 0 \end{bmatrix} \begin{bmatrix} \mathbf{a}^T & 0 \end{bmatrix} \tag{3}$$

where $\mathbf{0}$ is a $n \times 1$ vector of zeroes.

Let us denote the given rank-m SVD of the submatrix $K_n \approx U_{nm} \Sigma_m U_{nm}^T$. We then simply substitute this in (3) and rewrite the expression with block matrices:

$$K_{n+1} \approx \begin{bmatrix} U_{nm} \\ \mathbf{0}^T \end{bmatrix} \Sigma_m \begin{bmatrix} U_{nm}^T & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{a} \\ b \end{bmatrix} \frac{1}{b} \begin{bmatrix} \mathbf{a}^T & b \end{bmatrix} - \begin{bmatrix} \mathbf{a} \\ 0 \end{bmatrix} \frac{1}{b} \begin{bmatrix} \mathbf{a}^T & 0 \end{bmatrix}. \tag{4}$$

We can then organize the separate vectors altogether in a convenient symmetric factorization:

$$K_{n+1} \approx \left[\begin{array}{cc} U_0 & A \end{array}\right] \left[\begin{array}{cc} \Sigma_m & O \\ O^T & D \end{array}\right] \left[\begin{array}{c} U_0^T \\ A^T \end{array}\right] \tag{5}$$

in which O represents the $n \times 2$ matrix of zeroes, and in which we denote

$$U_0 := \left[\begin{array}{c} U_{nm} \\ \mathbf{0} \end{array}\right]. \tag{6}$$

The $(n+1) \times 2$ matrix A contains the extra columns,

$$A := \left[\begin{array}{cc} \mathbf{a} & \mathbf{a} \\ b & 0 \end{array}\right], \tag{7}$$

and the diagonal 2×2 matrix D contains the coefficients of the rank-1 updates,

$$D_u := \left[\begin{array}{cc} 1/b & 0 \\ 0 & -1/b \end{array}\right]. \tag{8}$$

We have obtained now an eigenspace, with a set of orthonormal eigenvectors in U_0, extended with the two columns of A:

$$\left[\begin{array}{cc} U_0 & A \end{array}\right]. \tag{9}$$

The two column vectors of A disturb the orthogonality of the above matrix. To restore this orthogonality, we need to compute the part of A that is orthogonal to the eigenspace. So we must decompose the vectors of A into a component $A_{U_0^\perp}$ orthogonal and a component A_{U_0} parallel to the U_0:

$$A = A_{U_0^\perp} + A_{U_0} = (I_n - U_0 U_0^T)A + U_0 U_0^T A. \tag{10}$$

This splits up the extending contribution of A into an orthogonal component that increases the rank of the eigenspace, while the parallel component will cause the eigenvectors needing to be rotated. This leads to a factorization

$$\left[\begin{array}{cc} U_0 & A \end{array}\right] = \left[\begin{array}{cc} U_0 & (I_n - U_0 U_0^T)A \end{array}\right] \left[\begin{array}{cc} I_n & U_0^T A \\ O^T & I_2 \end{array}\right]. \tag{11}$$

To make the column vectors of $A_{U_0^\perp}$ mutually orthogonal, we compute the QR-decomposition $Q_A R_A \xleftarrow{\text{QR}} A_{U_0^\perp}$, in which $A_{U_0^\perp}$ is only of dimension $(n+1) \times 2$. This yields the QR decomposition

$$\left[\begin{array}{cc} U_0 & A \end{array}\right] = \left[\begin{array}{cc} U_0 & Q_A \end{array}\right] \left[\begin{array}{cc} I_n & U_0^T A \\ O^T & R_A \end{array}\right] =: Q_u R_u. \tag{12}$$

So far, only $O(nm)$ operations are needed. In comparison to computing the QR factorization (12) from scratch, the number of computations has been limited by exploiting the mutual orthonormality of the columns of U_0. Substitution into (5) yields

$$K_{n+1} \approx \left[\begin{array}{cc} U_0 & Q_A \end{array}\right] \left[\begin{array}{cc} I_n & U_0^T A \\ O^T & R_A \end{array}\right] \left[\begin{array}{cc} \Sigma_m & O \\ O^T & D \end{array}\right] \left[\begin{array}{cc} I_n & O \\ A^T U_0 & R_A \end{array}\right] \left[\begin{array}{c} U_0^T \\ Q_A^T \end{array}\right]. \tag{13}$$

In order to obtain the optimal eigenvectors, a small SVD on the three middle matrices is necessary. The smallest singular value and corresponding eigenvector will then be discarded, which costs in total only a negligible $O(m^3)$ operations:

$$K_{n+1} \approx Q_u \left(R_u \left[\begin{array}{cc} \Sigma_m & O \\ O^T & D_u \end{array}\right] R_u^T \right) Q_u^T \overset{SVD}{=} Q_u V' \Sigma'_m V'^T Q_u^T, \tag{14}$$

It takes then finally $O(nm^2)$ operations to obtain an updated eigenspace $U' = QV'$, which is spent on the rotation of the orthonormal eigenspace Q_u according to V'. When the updating is followed by a downdating and downsizing, this last expensive operation may be postponed untill the end of the iteration step.

B. Downdating

For downdating, thus excluding the influence of a point, usually a forgetting factor is introduced for the previous eigenspace. This has on one hand the advantage to smooth data signal variations, but on the other hand it is only suitable for slowly varying processes. It is often more desirable to have a faster tracking response that can catch sudden signal changes, which requires that the net contribution of a point can be excluded.

The above key element in eq. (3) of the updating scheme was that the expanded matrix can be written in terms of a stripped matrix to which a difference of 2 rank-one matrices is added. It gives in fact immediately the right tool for performing a downdate, because in this case we only have other rank-one matrices, since a row and column are now stripped:

$$\begin{bmatrix} 0 & 0 \\ 0^T & K_n \end{bmatrix} = \begin{bmatrix} d & \mathbf{c}^T \\ \mathbf{c} & K_n \end{bmatrix} - \frac{1}{d} \begin{bmatrix} d \\ \mathbf{c} \end{bmatrix} \begin{bmatrix} d & \mathbf{c}^T \end{bmatrix} + \frac{1}{d} \begin{bmatrix} 0 \\ \mathbf{c} \end{bmatrix} \begin{bmatrix} 0 & \mathbf{c}^T \end{bmatrix}. \tag{15}$$

We do not repeat the procedure for obtaining the eigenspace because it is completely analogous with the updating, only the matrices are slightly different.

There is however one difference concerning dimensions here. Whereas the row dimension of the eigenspace for an update grows from n to $n+1$, the reverse is not the case for a downdate. If the kernel matrix is exactly rank-n, then the first row of U' will be zero. However, in practice this will not be exactly the case. In next section we propose a downsizing scheme to reduce the dimension of the kernel matrix.

III. Downsizing

As mentioned in the previous section, for each update the row dimension of the eigenspace matrix increases. This is not practical for on-line applications where one wants a subspace tracker, where the dimension of the eigenspace is to remain constant. This means we must downsize the matrix after each update, thereby preserving orthogonality. In this section we propose an efficient scheme to accomplish that and recover the orthogonal eigenspace, up to a unitary transformation.

In downsizing, the first row of U', denoted by the column vector v, must be deleted, such that we can continue with the resulting leftover part V:

$$U' = \begin{bmatrix} v^T \\ V \end{bmatrix} \tag{16}$$

In practice, the columns of V are not exactly mutually orthonormal. Orthonormality may be restored by means of a QR-decomposition of V, but this is expensive. In this section we will present a cheap approximate solution.

Consider

$$U'^T U' = I_m = \begin{bmatrix} v & V^T \end{bmatrix} \begin{bmatrix} v^T \\ V \end{bmatrix} = V^T V + \|v\|^2 \bar{v}\bar{v}^T \tag{17}$$

where $\bar{v} = v/\|v\|$ is vector v divided by its length.

Basically, we need a transformation matrix M such that V will be orthogonalized:

$$VM = Q \qquad \text{with} \qquad Q^T Q = I_m. \tag{18}$$

In this orthogonality condition we substitute (17) which yields

$$I_m = M^T V^T V M = M^T (I_m - \|v\|^2 \bar{v}\bar{v}^T) M. \tag{19}$$

Exploiting the structure, a convenient choice for M is

$$M = \begin{bmatrix} \bar{v}^T & \bar{v}_\perp \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{1-\|v\|^2}} & 0 \\ 0 & I_m \end{bmatrix} =: CJ, \tag{20}$$

where $\bar{v}_\perp$ is the $r \times (r-1)$ orthogonal complement of $\bar{v}$ such that $C^TC = I = CC^T$. The computation of the matrix $\bar{v}_\perp$ costs a negligible $O(m^3)$ operations since it is the right nullspace of $\bar{v}^T$ (in Matlab implemented as null($\bar{v}^T$)).

To conclude, the $(n+1) \times m$ eigenspace U' is replaced by the $n \times m$ eigenspace Q:

$$U' = \begin{bmatrix} v^T \\ V \end{bmatrix} \xrightarrow{\text{downsize}} Q := VM. \tag{21}$$

The total downsizing cost is dominated by this transformation itself, where M is left multiplied with V, which costs $O(nm^2)$ operations.

Of course the matrix with singular values is accordingly adapted to absorb the effect of the transformation matrix:

$$\Sigma'_m \rightarrow J^{-1}C^T\Sigma'_mC(J^{-1})^T, \tag{22}$$

which has only a negligible operation cost of $O(m^3)$ operations. If the downsizing is preceded by an update and downdate, the SVD of those steps may both be postponed until after downsizing. In this way only one SVD per iteration step is required for tracking.

The schemes of updating and downdating form in combination with this downsizing method a fast dominant eigenspace tracker algorithm that needs per step only $O(nm^2)$ operations and $O(nm)$ memory space. The overall algorithm is summarized in Table I.

$K_n^{(k)}$ is a kernel matrix associated with data points $\mathbf{x}_k, \mathbf{x}_{k+1}, \ldots, \mathbf{x}_{k+n-1}$.
Given $U_{nm}^{(k)}\ \Sigma_m^{(k)}\ (U_{nm}^{(k)})^T \xleftarrow[m\text{-rank}]{\text{SVD}} K_n$.
$K_n^{(k+1)}$ is a kernel matrix associated with data points $\mathbf{x}_{k+1}, \mathbf{x}_{k+2}, \ldots, \mathbf{x}_{k+n}$.
Calculate $K_n^{(k+1)} \approx U_{nm}^{(k+1)}\ \Sigma_m^{(k+1)}\ (U_{nm}^{(k+1)})^T$.

- Update:
  - $U_0 = \begin{bmatrix} U_{nm}^{(k)} \\ \mathbf{0} \end{bmatrix}$
  - $Q_AR_A \xleftarrow{\text{QR}} (I - U_0U_0^T) \begin{bmatrix} \mathbf{a} & \mathbf{a} \\ b & 0 \end{bmatrix}$
  - $Q_u = [U_0\ Q_A]$
  - $\Sigma_u = R_u \begin{bmatrix} \Sigma_m^{(k)} & O \\ O^T & D_u \end{bmatrix} R_u^T$
- Downdate:
  - $Q_BR_B \xleftarrow{\text{QR}} (I - Q_uQ_u^T) \begin{bmatrix} d & 0 \\ \mathbf{c} & \mathbf{c} \end{bmatrix}$
  - $Q_d = [Q_u\ Q_B]$
  - $\Sigma_d = R_d \begin{bmatrix} \Sigma_u & O \\ O^T & D_d \end{bmatrix} R_d^T$
- Downsize: $Q_d \rightarrow$ orthogonal eigenspace T and transformation M
- Perform SVD:
  - $W\ \Sigma_m^{(k+1)}\ W^T \xleftarrow{\text{SVD}} M^T\Sigma_dM$
  - discard last 4 singular values and vectors
- Perform rotation: $U_{nm}^{(k+1)} = TW$

TABLE I

SCHEMATIC OVERVIEW OF THE SUBSPACE TRACKING ALGORITHM.

IV. EXPERIMENTS

We implemented the proposed up/downdating and downsizing schemes in Matlab. A naive comparison with the Matlab SVDS routine shows that computational gain is indeed achieved, especially for large dimensions and few eigenvectors. Yet, in the experiments we aim primarily to characterize how well the true eigenvectors are approximated whilst tracking (ie up/downdating and downsizing) a kernel matrix.

We consider a representative example of a kernel matrix, based upon the known Abalone benchmark dataset [14] with $n = 3000$ training instances having dimension $p = 7$. We consider the common choice of the radial basis kernel function (with h the kernel width parameter, fixed at 18.57):

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{h^2}\right), \tag{23}$$

A. Eigenspace Tracking

We track of this kernel matrix the eigenspace, of dimension 500×8. We take some extra margin and track the 20 first eigenvectors to compare to the batch SVD. In figure (1) the eigenspectrum of K is shown, which has a quite steep exponentially decreasing behaviour, typical in most kernel applications.

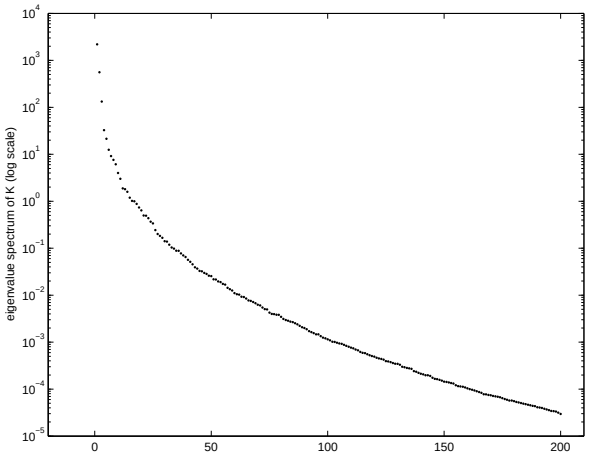


Fig. 1. Eigenspectrum of the symmetric K matrix, shown for the first 300 (of 3000) values.

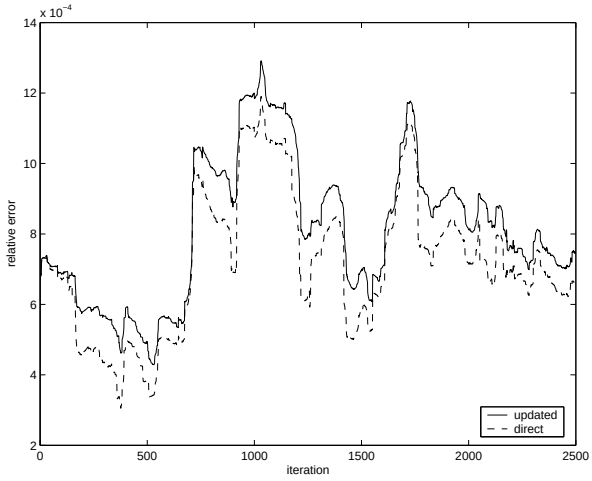


Fig. 2. Relative error versus iteration step, shown for the updated eigenspace and the best-rank eigenspace obtained by a batch SVD. The two plot lines follow each other closely and indicate good approximation of the eigenspace.

In order to assess accuracy and stability of the tracking algorithm, we employed the following measures.

- A good low-rank approximation delivers a faithful representation of the original matrix. Therefore, an overall measure of accuracy often used is the relative error:

$$\frac{\|K - \tilde{K}\|_F}{\|K\|_F}, \tag{24}$$

where $\tilde{K}$ is a rank-m approximation of K and $\|\cdot\|_F$ denotes the Frobenius norm. In this case we have a low-rank approximation $\tilde{K} = \tilde{U}\tilde{S}\tilde{U}^T$ where $\tilde{U}$ is the eigenspace of K and $\tilde{S}$ is the matrix with singular values on the diagonal, both resulting from our algorithm. In our experiments we compare the relative error of the rank-m approximation obtained by our algorithm and the best m-rank approximation obtained by the batch SVD. Figure (2) shows for the example that the tracked eigenspace approximates the batch eigenspace very closely and consistently.

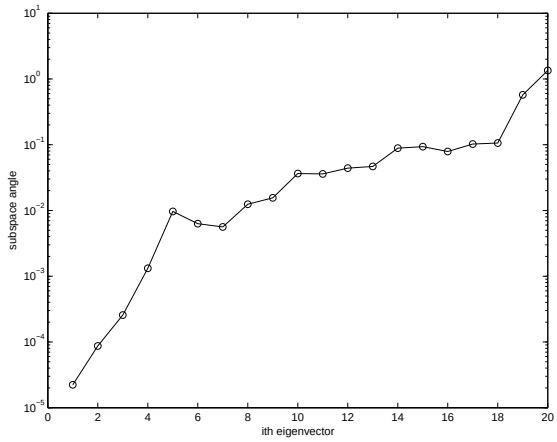


Fig. 3. Subspace angle for each eigenvector. The dominant eigenspace is still being well-approximated after 2500 iteration steps.

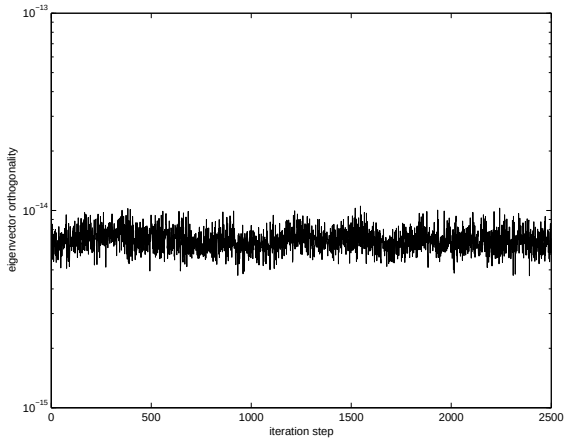


Fig. 4. Orthogonality of the eigenvectors versus iteration step. Smaller values indicate better preservation of mutual orthogonality between the eigenvectors.

- The subspace angles [15] between the dominant eigenspaces of $\tilde{K}$ and K are direct indicators of how close these eigenspaces are. Figure (3) shows for the first eigenvectors eigenvector the subspace angle, which can range from zero to $arcsin(1) = 1.5708$. The approximation is found to be closely connected to the gap between the successive eigenvalues. From the moment the eigenspectrum becomes flat (between eigenvalue 5 and 10), the corresponding errors become larger.
- Another important criterion concerns the orthogonality of the eigenspace during successive iterations. This can be monitored by the scalar orthogonality measure

$$\|I - \tilde{U}^T \tilde{U}\|_2. \tag{25}$$

  A zero value indicates perfect orthogonality. In figure (4) we show the typical evolution of this measure per iterative step. In all experiments the orthogonality is consistently very well-preserved and shows values in the range $[10^{-13}, 10^{-15}]$.

We conclude that, given the fact that the spectrum is continuous, the results are as accurate as can be expected.

B. Updating

If we consider solely updating (without downdating or downsizing) we expect also good convergence to the real eigenspace. Table II shows relative errors, defined in expr. (24) and their ratios for different initial eigenbasis dimensions.

n	m	relative error (updating)	relative error (batch)	ratio
500	20	0.0133082	0.0132063	1.0077
500	40	0.0050548	0.0047341	1.0677
1000	20	0.0132701	0.0132063	1.0048
1000	40	0.0050505	0.0047341	1.0668
1000	60	0.0025403	0.0024522	1.0359
1000	100	0.0008069	0.0007887	1.0230

TABLE II

RELATIVE ERRORS FOR THE m-RANK APPROXIMATIONS OF THE FULL KERNEL MATRIX, OBTAINED BY THE SVD UPDATING ALGORITHM AND THE BATCH SVD. THE CLOSE TO ONE RATIOS INDICATE GOOD PRESERVATION OF ACCURACY.

From the ratio we see that the difference can vary up to a few percent. Orthogonality was on average preserved up to values of 10^{-14}. Analogously the other measures demonstrate a good to excellent level of numerical precision for the obtained eigenspace, which is related to the gap between the eigenvalues.

V. Conclusions

This paper introduced a novel up- and downdating algorithm for the dominant eigenspace of a large scale symmetric square matrix, in which the adaptation simultaneously occurs both in the row and columns. Additionally a downsizing mechanism was proposed such that the algorithm is also capable of tracking the dominant eigenspace.

Kernel based methods in machine learning [2] are the vast context in which this kind of matrix is prevalent. It is its dominant eigenspace that is frequently needed in the solving of problems of classification, pattern recognition, function estimation, system identification and signal processing.

Although kernel methods are powerful solvers because they are able to deal with nonlinearity, they are severely limited when it comes to large scale applications. In that case the computation times become unacceptable or the kernel matrix cannot even be completely stored in memory. Further there is no true on-line kernel principal component analysis algorithm known that can deal with time-varying data. Our algorithm fills this gap, in an efficient manner with a time complexity of $O(nm^2)$ and memory requirements of $O(nm)$ per iteration.

In some first experiments we focused on the accuracy and stability of the algorithm and found very good to excellent approximation of the eigenvectors with preservation of the orthogonality and small relative error for the resulting low-rank approximations.

Acknowledgments

This research work was carried out at the ESAT laboratory of the Katholieke Universiteit Leuven. It is supported by grants from several funding agencies and sources: Research Council KU Leuven: Concerted Research Action GOA-Mefisto-666 (Mathematical Engineering), IDO (IOTA Oncology, Genetic networks), PhD/postdoc & fellow grants; Flemish Government: Fund for Scientific Research Flanders (PhD/postdoc grants, projects G.0407.02 (support vector machines), G.0256.97 (subspace), G.0115.01 (bio-i & microarrays), G.0240.99 (multilinear algebra), G.0197.02 (power islands), research communities ICCoS, ANMMM), AWI (Bil. Int. Collaboration Hungary/ Poland), IWT (Soft4s (softsensors), STWW-Genprom (gene promotor prediction), GBOU-McKnow (Knowledge management algorithms), Eureka-Impact (MPC-control), Eureka-FLiTE (flutter modeling), PhD grants); Belgian Federal Government: DWTC (IUAP IV-02 (1996-2001) and IUAP V-10-29 (2002-2006) and IUAP V-22 (2002-2006): Dynamical Systems & Control: Computation, Identification & Modelling), Program Sustainable Development PODO-II (CP/40: Sustainibility effects of Traffic Management Systems); Direct contract research: Verhaert, Electrabel, Elia, Data4s, IPCOS. Luc Hoegaerts is a PhD student supported by the Flemish Institute for the Promotion of Scientific and Technological Research in the Industry (IWT). Lieven De Lathauwer holds a permanent research position with the French Centre National de la Recherche Scientifique (C.N.R.S.); he also holds a honorary research position with the K.U.Leuven, Leuven, Belgium. Johan Suykens is an associate professor at KU Leuven, Belgium. Joos Vandewalle is a full professor at KU Leuven, Belgium.

References

[1] B. Schölkopf and A. Smola, *Learning with kernels*. MIT Press, 2002.
[2] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machines*. World Scientific, Singapore, 2002.
[3] G. Wahba, *Spline Models for Observational Data*, ser. CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia, PA: SIAM, 1990, vol. 59.
[4] B. Schölkopf, A. Smola, and K. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10(5), pp. 1299–1319, 1998.
[5] R. Rosipal and M. Girolami, "An expectation-maximization approach to nonlinear component analysis," *Neural Computation*, vol. 13, no. 3, pp. 505–510, 2001.
[6] K. I. Kim, M. Franz, and B. Schlkopf, "Kernel hebbian algorithm for iterative kernel principal component analysis," Max-Planck-Institut fr biologische Kybernetik, Tbingen, Tech. Rep. 109, June 2003.
[7] A. Kuh, "Adaptive kernel methods for cdma systems," in *Proc. of the International Joint Conference on Neural Networks (IJCNN 2001)*, Washington DC, 2001, pp. 1404–1409.
[8] B. P., "Updating a singular value decomposition," *BIT*, vol. 10, pp. 376–385, 1970.
[9] J. R. Bunch and N. J. P., "Updating the singular value decomposition," *Numer. Math.*, vol. 31, pp. 111–129, 1978.
[10] M. Moonen, P. V. Dooren, and J. Vandewalle, "An svd updating algorithm for subspace tracking," *SIAM J. Matrix Anal. Appl.*, vol. 13, no. 4, pp. 1015–1038, 1992.
[11] S. Chandrasekaran, B. S. Manjunath, Y. F. Wang, J. Winkeler, and H. Zhang, "An eigenspace update algorithm for image analysis," *Graphical models and image processing: GMIP*, vol. 59, no. 5, pp. 321–332, 1997.
[12] A. Levy and M. Lindenbaum, "Sequential karhunen-loeve basis extraction," *IEEE Transactions on Image processing*, vol. 9, no. 8, pp. 1371–1374, 2000.
[13] R. Badeau, G. Richard, and B. David, "Sliding window adaptive svd algorithms," *IEEE Transactions on Signal Processing*, vol. 52, no. 1, janvier 2004.
[14] C. Blake and C. Merz, "Uci repository of machine learning databases," 1998. [Online]. Available: http://www.ics.uci.edu/∼mlearn/-MLRepository.html
[15] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 2nd ed. Baltimore, Maryland 21218: The John Hopkins University Press, 1989.