Adaptive Transform Coding Using LMS-like Principal Component Tracking

Vivek K Goyal[1] and Martin Vetterli[1,2]

[1]Dept. of Electrical Eng. & Comp. Sci.
University of California, Berkeley
Berkeley, CA, USA

[2]Laboratoire de Communications Audiovisuelles
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland

Abstract

A new set of algorithms for transform adaptation in adaptive transform coding is presented. These algorithms are inspired by standard techniques in adaptive finite impulse response (FIR) Wiener filtering and demonstrate that similar algorithms with simple updates exist for tracking principal components (eigenvectors of a correlation matrix). For coding an N-dimensional source, the transform adaptation problem is posed as an unconstrained minimization over $K = N(N-1)/2$ parameters, and this for two possible performance measures. Performing this minimization through a gradient descent gives an algorithm analogous to LMS. Step size bounds for stability similar in form to those for LMS are proven. Linear and fixed-step random search methods are also considered. The stochastic gradient descent algorithm is simulated for both time-invariant and slowly-varying sources. A "backward-adaptive" mode, where the adaptation is based on quantized data so that the decoder and encoder can maintain the same state without side information, is also considered.

EDICS Category: SP 2. Digital Signal Processing
SP 2.6 Adaptive Filters
SP 2.6.2 Algorithms (Gradient-Descent, RLS, Random Search, Genetic)

Corresponding author: Vivek K Goyal
211-79 Cory Hall #1772
Berkeley, CA 94720-1772
e-mail: v.goyal@ieee.org
voice: +1 510 643 5798
fax: +1 510 642 2845

I. Introduction

Optimal linear transform coding has two striking similarities with optimal finite impulse response (FIR) Wiener filtering: both (often unrealistically) require knowledge of second-order moments of signals; and both require a calculation which is considered expensive if it must be done repeatedly (eigendecomposition and matrix inversion, respectively). In FIR Wiener filtering, it is well-known that these difficulties can be mitigated by adaptation. This paper establishes new methods in block transform adaptation that are analagous to some of the standard methods in adaptive FIR Wiener filtering.

The basis for many adaptive Wiener filtering methods is to specify independent parameters, define a performance surface with respect to these parameters, and to search the performance surface for the optimal parameter values. The most common method of performance surface search is gradient descent—which leads to the LMS algorithm [1]—but linear and fixed-step random searches [2] also fall into this class. This paper defines two meaningful performance surfaces (cost functions) for linear transform coding and analyzes various search methods for these surfaces. The result is a set of new algorithms for adaptive linear transform coding.

Subject to a Gaussian condition on the source and fine-quantization approximations,[1] finding an optimal transform for transform coding amounts to finding an orthonormal set of eigenvectors of a symmetric, positive semidefinite matrix; *i.e.*, finding an optimal transform is an instance of the *symmetric eigenproblem*, a fundamental problem of numerical analysis [3]. Thus, in finding a method for transform adaptation we are in fact attempting to approximately solve a sequence of symmetric eigenvalue problems. The idea of using performance surface search (*i.e.*, cost function minimization) for this problem seems to be new, although the cost function which we will later call J_1 has been used in convergence analyses [3]. The algorithms we develop here are *not* competitive with cyclic Jacobi methods for computing a *single* eigendecomposition of a large matrix; however, they are potentially useful for computing eigendecompositions of a slowly varying sequence of matrices.

The novelty and potential utility of these algorithms for transform coding comes from the following properties: the transform is always represented by a minimal number of parameters, the autocorrelation matrix of the source need not be explicitly estimated, and the computations

[1]Without these technical conditions, there is no general principle for determining the optimal transform, so in the remainder of the paper we revert to using "optimal" without qualification. For more details see Appendix A.

are more parallelizable than cyclic Jacobi methods. In addition, further insights may come from drawing together techniques from adaptive filtering, transform coding, and numerical linear algebra.

The reader is referred to [3] for a thorough treatment of the techniques for computing eigendecompositions including the techniques specific to the common special case where the matrix is symmetric. Appendices A and B provide brief reviews of transform coding and adaptive FIR Wiener filtering, respectively.

II. Problem Definition, Basic Strategy, and Outline

Let $\{x_n\}_{n\in\mathbb{Z}^+}$ be a sequence of $\mathbb{R}^N$-valued random vectors and let $X_n = E[x_n x_n^T]$. We assume that the dependence of X_n on n is mild[2] and desire a procedure which produces a sequence of orthogonal transforms T_n such that $Y_n = T_n X_n T_n^T$ is approximately diagonal for each n. The procedure should be causal, *i.e.*, T_k should depend only on $\{x_n\}_{n=1}^k$. X_n will not be known, but must be estimated or in some sense inferred from $\{x_k\}_{k=1}^n$.

First of all, note that if X_n is known, then a T_n consisting of normalized eigenvectors of X_n solves our problem [8]. A traditional approach would be to construct an estimate $\hat{X}_n = f(\{x_k\}_{k=1}^n)$ for each n, and then use an "off the shelf" method to compute the eigenvectors of $\hat{X}_n$. The difficulty with this is that the eigenvector computation may be deemed too complex to be done for each n.

In analogy to the way the LMS algorithm avoids explicitly solving a linear system of equations (see Appendix B), we wish to avoid using an explicit eigendecomposition algorithm. The first conceptual step is to replace the problem of finding a diagonalizing transform T_n for X_n with a minimization problem for which a diagonalizing transform achieves the minimum. The next step is to derive a gradient descent iteration for the minimization problem. Note that in these two steps we assume that X_n is known. The final step is to apply the gradient descent iteration with X_n replaced by a stochastic approximation $\hat{X}_n$. The following three sections address these three steps. In Section III we give two cost functions which are minimized by a diagonalizing transform. Section IV gives derivations for gradient descents with respect to the two cost functions along with step size bounds which ensure local convergence. Linear and fixed-step random searches are

[2]If the dependence of X_n on n is not mild, then it is rather hopeless to use adaptation in the traditional sense of learning source behavior based on the recent past. Better strategies might include classification [4], [5] or other basis selection methods [6], [7].

also discussed. Section IV contains the linear algebraic computations which underlie the signal processing algorithms which are ultimately presented in Section V. It is in this final section that we stochastically simulate applications to adaptive transform coding.

III. Performance Criteria

If two orthogonal transforms only approximately diagonalize X, which of the two is better? In order to use a performance surface search to iteratively find optimal transforms, we need a continuous measure of the diagonalizing performance of a transform. The remainder of the paper uses two such performance measures.

The most obvious choice for a cost function is the squared norm of the off-diagonal elements of $Y = TXT^T$:

$$J_1(T) = \sum_{i \neq j} Y_{ij}^2 \tag{1}$$

This cost function is clearly nonnegative and continuous in each component of T. Also, $J_1(T) = 0$ if and only if T exactly diagonalizes X.

The cost function

$$J_2(T) = \prod_{i=1}^{N} Y_{ii}. \tag{2}$$

is intimately connected to transform coding theory but is less obviously connected to the diagonalization of X. Under the standard assumptions of transform coding, for a fixed rate, $\sqrt[N]{J_2(T)}$ is proportional to the distortion (see Appendix A, (18)). Thus minimizing J_2 minimizes the distortion and $J_2(T)$ is minimized by the transform which diagonalizes X. A potential disadvantage of this cost function is that the minimum value is not zero; instead it is $\prod_i \lambda_i$, where λ_i's are the eigenvalues of X.

IV. Methods for Performance Surface Search

In Section IV-C we present two new eigendecomposition algorithms based on gradient descent with respect to the cost functions J_1 and J_2. These algorithms and the random search algorithm of Section IV-B are inspired by and parallel the standard methods in adaptive FIR Wiener filtering [2]. For comparison, standard methods which are computationally attractive for computing single eigendecompositions are presented in Section IV-D.

The effects of the time variation of X and estimation noise are left for subsequent sections. Hence, throughout this section we dispense with time indices and consider iterative methods for

diagonalizing a fixed matrix X.

A. Parameterization of Transform Matrices

An $N \times N$ orthogonal matrix has fewer than N^2 independent parameters because of the requirement that the columns (or equivalently the rows) form an orthonormal set. In our search for the best orthogonal transform it will sometimes be useful to represent the matrix in terms of the smallest possible number of parameters.

To determine the number of degrees of freedom in the parameterization of an orthogonal matrix, imagine that one is constructing such a matrix column-by-column. Making the ith column orthogonal to the earlier columns leaves $N - i + 1$ degrees of freedom and normalizing gives $N - i$ degrees of freedom plus a choice of sign. Thus overall there are $N(N-1)/2$ degrees of freedom plus N sign choices. The sign choices have no effect on $J_1(T)$ or $J_2(T)$, so we are left with $K = N(N-1)/2$ degrees of freedom. $K = \binom{N}{2}$ matches the number of distinct Givens rotations, and we will see in Lemma 1 below that the parameters of interest can be taken to be the angles of Givens rotations.

Definition 1: A matrix of the form

$$\widetilde{G}_{i,j,\theta} = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos\theta & \cdots & \sin\theta & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -\sin\theta & \cdots & \cos\theta & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \\ \begin{array}{ccccccc} & & i & & j & & \end{array} \end{array} \begin{array}{l} \\ \\ i \\ \\ j \\ \\ \\ \end{array} , \tag{3}$$

where $-\pi/2 < \theta \leq \pi/2$, is called a *Givens (or Jacobi) rotation* [3]. It can be interpreted as a counterclockwise rotation of θ radians in the (i, j) coordinate plane.

Since we will be interested in Givens rotations with $i < j$, it will be convenient to use the index remapping $G_{k,\theta} = \tilde{G}_{i,j,\theta}$, where (i, j) is the kth entry of a lexicographical list of $(i, j) \in \{1, 2, \ldots, N\}^2$ pairs with $i < j$. For example, the matrix below gives the corresponding value of

k in the (i,j) location for $N = 4$:

$$\begin{bmatrix} \star & 1 & 2 & 3 \\ \star & \star & 4 & 5 \\ \star & \star & \star & 6 \\ \star & \star & \star & \star \end{bmatrix}$$

Lemma 1: Let $X \in \mathbb{R}^{N\times N}$ be a symmetric and let $K = N(N-1)/2$. Then there exists $\Theta = [\theta_1, \theta_2, \ldots, \theta_K]^T \in [-\pi/2, \pi/2)^K$ such that $T_\Theta X T_\Theta^T$ is diagonal, where

$$T_\Theta = G_{1,\theta_1} G_{2,\theta_2} \ldots G_{K,\theta_K}. \tag{4}$$

Proof: Since X is symmetric, there exists an orthogonal matrix S such that SXS^T is diagonal [8]. Any orthogonal matrix can be factored as

$$S = (\widetilde{G}_{1,2,\theta_{1,2}} \widetilde{G}_{1,3,\theta_{1,3}} \cdots \widetilde{G}_{1,N,\theta_{1,N}})(\widetilde{G}_{2,3,\theta_{2,3}} \cdots \widetilde{G}_{2,N,\theta_{2,N}}) \cdots (\widetilde{G}_{N-1,N,\theta_{N-1,N}}) D_\epsilon,$$

where $D_\epsilon = \mathrm{diag}(\epsilon_1, \ldots, \epsilon_N)$, $\epsilon_i = \pm 1$, $i = 1, 2, \ldots, N$ [9]. It is now obvious that we can take $T = SD_\epsilon^{-1}$ because $D_\epsilon^{-1} X (D_\epsilon^{-1})^T = X$. ■

B. Random Search

In light of Lemma 1 and the discussion of Section III, finding a diagonalizing transform amounts to minimizing J_1 or J_2 (written as J where either fits equally) over $\Theta \in [-\pi/2, \pi/2)^K$. Conceptually, the simplest way to minimize a function—so simple and naive that it is often excluded from consideration—is to guess.

We could discretize the range of interest of Θ, evaluate J at each point on the grid, and take the minimum of these as an approximation to the minimum. The accuracy of this approximation will depend on the smoothness of J and the density of the grid. The grid could also be made adaptive to have higher density of points where J is smaller. This exhaustive deterministic approach is not well suited to our application with a slowly-varying sequence of X matrices because information from previous iterations is not easily incorporated. Instead, we present two approaches which yield a random sequence of parameter vectors with expected drift toward the optimum.

In a *fixed-step random search*, a small random change is tentatively added to the parameter vector. The change is adopted if it decreases the objective function; else, it is discarded. Formally, the update is described by

$$\Theta_{k+1} = \begin{cases} \Theta_k + \alpha\eta_k & \text{if } J(\Theta_k + \alpha\eta_k) < J(\Theta_k), \\ \Theta_k & \text{otherwise,} \end{cases}$$

where $\alpha \in \mathbb{R}^+$ and $E[\eta_k \eta_k^T] = I$.

A fixed-step random search makes no progress on an iteration where $\Theta_k + \alpha\eta_k$ is found to be worse than Θ_k. Another possibility is a *linear random search* [2]. In this case, instead of taking no step if η_k seems to be a step in the wrong direction, one takes a step in the opposite direction; the size of each step is proportional to the increase or decrease in J. The update is described by

$$\Theta_{k+1} = \Theta_k + \alpha[J(\Theta_k) - J(\Theta_k + \sigma\eta_k)]\eta_k,$$

where $\alpha, \sigma \in \mathbb{R}^+$ and $E[\eta_k \eta_k^T] = I$.

It is intuitively clear that, using either cost function, for sufficiently small α both random search algorithms tend to drift toward local minima of J. The fixed-step and linear random search algorithms were simulated on the problem $X = \mathrm{diag}([1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}])$ with initial guess Θ_0 chosen randomly according to a uniform distribution on $[-\pi/2, \pi/2]^6$. Figure 1 gives the averaged results of 400 simulations of 400 iterations each for various values of α. Gaussian η was used for the fixed-step search; for the linear search, η is uniformly distributed on the unit sphere and $\sigma = 0.01$.

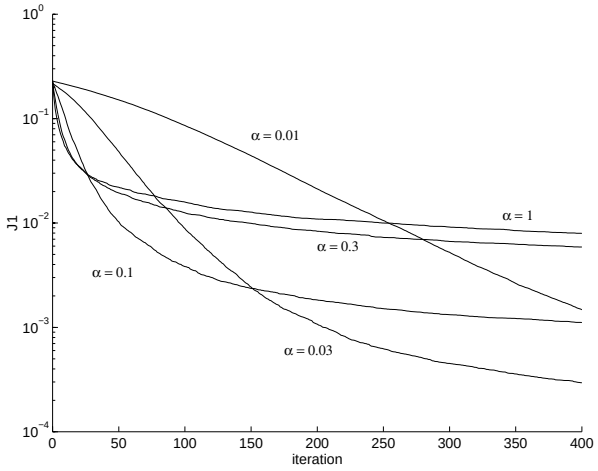
As shown in Figure 1(a)–(b), the fixed-step searches have the undesirable quality that the best choice of α depends on the number of iterations: for a small number of iterations a large α is preferred while for a large number of iterations the opposite is true. A simple interpretation of this is that for large α the first few steps are more beneficial, but as the optimum Θ is approached, tentative steps are very unlikely to be accepted; close to the optimum Θ, small α is more likely to yield improvements.

While the fixed-step algorithm tends to get stuck when α is large, the performance of the linear search algorithm degrades in a different way. When α is large, many steps are taken which increase J; hence the convergence gets more erratic. For very large α there is no negative drift in J. This is shown in Figure 1(c)–(d).

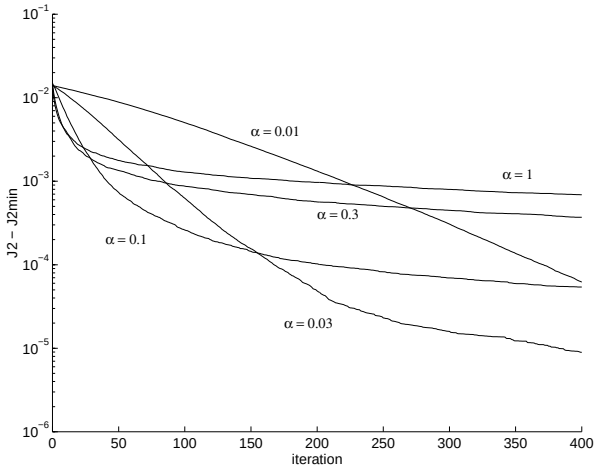
The conceptual simplicity of random search algorithms comes from utilizing no knowledge of the function to be minimized. Using gradient descent is one way to utilize knowledge of the function to be minimized. This is discussed in the following section.

C. Descent Methods

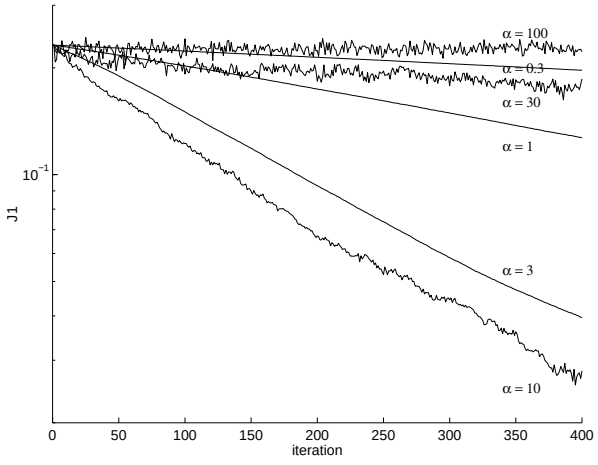
In this section we will explore gradient descent based methods for minimizing J_1 or J_2. The idea of a gradient descent is very simple. Suppose we wish to find Θ which minimizes a function



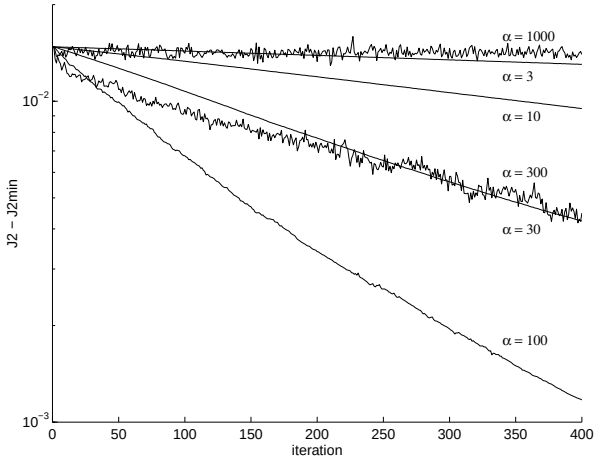
(a) Fixed-step search with respect to J_1



(b) Fixed-step search with respect to J_2



(c) Linear search with respect to J_1



(d) Linear search with respect to J_2

Fig. 1. Simulations of the random search algorithms. $X = \mathrm{diag}([1,\ \frac{1}{2},\ \frac{1}{4},\ \frac{1}{8}])$ and results are averaged over 400 randomly chosen initial conditions Θ_0.

$J(\Theta)$ and we have an initial guess Θ_0. Assuming a first-order approximation of J, changing Θ_0 in the direction of $\nabla J|_{\Theta=\Theta_0}$ produces the maximum increase in J, so taking a step in the opposite direction produces the maximum decrease in J. This leads to the general update formula for gradient descent:

$$\Theta_{k+1} = \Theta_k - \alpha \, \nabla J|_{\Theta=\Theta_k} , \tag{5}$$

where $\alpha \in \mathbb{R}^+$ is the step size. We now compute the gradient and the bounds on α for stability for each of the cost function of Section III.

C.1 Minimization of J_1

We start by computing ∇J_1 elementwise. Firstly,

$$\frac{\partial J_1}{\partial \theta_k} = \sum_{i \neq j} \frac{\partial}{\partial \theta_k} Y_{ij}^2 = \sum_{i \neq j} 2 Y_{ij} \frac{\partial Y_{ij}}{\partial \theta_k}. \tag{6}$$

For notational convenience, let $U_{(a,b)} = G_{a,\theta_a} G_{a+1,\theta_{a+1}} \ldots G_{b,\theta_b}$, where $U_{(a,b)} = I$ if $b < a$, $U_k = U_{(k,k)}$, and $V_k = \frac{\partial}{\partial \theta} U_{k,\theta_k}$. Define $A^{(k)}$, $1 \leq k \leq K$, elementwise by $A_{ij}^{(k)} = \partial Y_{ij} / \partial \theta_k$. Then to evaluate $\partial Y_{ij} / \partial \theta_k$, write $Y = TXT^T$ and use (4) to yield

$$A^{(k)} = U_{(1,k-1)} V_k U_{(k+1,K)} X U_{(1,K)}^T + U_{(1,K)} X U_{(k+1,K)}^T V_k^T U_{(1,k-1)}^T \tag{7}$$

Combining (6) and (7),

$$\frac{\partial J_1}{\partial \theta_k} = 2 \sum_{i \neq j} Y_{ij} A_{ij}^{(k)}. \tag{8}$$

Theorem 1: Denote the eigenvalues of X by $\lambda_1, \lambda_2, \ldots, \lambda_N$ and let $\Theta_\star$ correspond to a diagonalizing transform for X. Then for Θ_0 sufficiently close to $\Theta_\star$ the gradient descent algorithm described by (5) and (8) converges to $\Theta_\star$ if

$$0 < \alpha < \left[2 \max_{i,j} (\lambda_i - \lambda_j)^2 \right]^{-1}$$

Proof: Without loss of generality, we can assume that $X = \mathrm{diag}([\lambda_1 \ \lambda_2 \ \cdots \lambda_N])$. This amounts to selecting the coordinates such that $\Theta_\star = \mathbf{0}$.

The key to the proof is observing that (5) describes an autonomous, nonlinear, discrete-time dynamical system and linearizing the system. We write

$$\Theta_{k+1} = \Theta_k - \alpha f(\Theta_k),$$

which upon linearization about $\mathbf{0}$ gives

$$\hat{\Theta}_{k+1} = (I - \alpha F) \hat{\Theta}_k,$$

where $F_{ij} = \left[\frac{\partial}{\partial \theta_j} f_i(\Theta) \right]_{\Theta = \mathbf{0}}$. A sufficient condition for local convergence is that the eigenvalues of $I - \alpha F$ lie in the unit circle. The fact that the local exponential stability of the original nonlinear system can be inferred from an eigenvalue condition on the linearized system follows from the continuous differentiability of F [10].

We now evaluate F. Differentiating (8) gives

$$\frac{\partial^2 J_1}{\partial \theta_\ell \, \partial \theta_k} = 2 \sum_{i \neq j} \left(Y_{ij} \frac{\partial A_{ij}^{(k)}}{\partial \theta_\ell} + A_{ij}^{(k)} \frac{\partial Y_{ij}}{\partial \theta_\ell} \right)$$

$$= \quad 2\sum_{i\neq j}\left(Y_{ij}\frac{\partial A_{ij}^{(k)}}{\partial\theta_\ell}+A_{ij}^{(k)}A_{ij}^{(\ell)}\right). \tag{9}$$

Evaluating (9) at $\Theta=\mathbf{0}$, Y becomes X (diagonal), so the first term makes no contribution; we need not attempt to calculate $\frac{\partial A_{ij}^{(k)}}{\partial\theta_\ell}$. By inspection of (7), $A^{(k)}$ becomes $V_kX+XV_k^T$. This simplifies further to a matrix which is all zeros except for having $\lambda_{j_k}-\lambda_{i_k}$ in the (i_k,j_k) and (j_k,i_k) positions, where (i_k,j_k) is the (i,j) pair corresponding to k in the index remapping discussed following Definition 1. Noting now that $A^{(k)}$ and $A^{(\ell)}$ have nonzero entries in the same positions only if $k=\ell$, we are prepared to conclude that

$$F_{k\ell}=\left[\frac{\partial^2 J_1}{\partial\theta_\ell\,\partial\theta_k}\right]_{\Theta=\mathbf{0}}=\begin{cases}4(\lambda_{i_k}-\lambda_{j_k})^2 & \text{if } k=\ell,\\ 0 & \text{otherwise.}\end{cases} \tag{10}$$

The eigenvalues of $I-\alpha F$ are $1-4\alpha(\lambda_{i_k}-\lambda_{j_k})^2$. The proof is completed by requiring that these all lie in the unit circle. ■

The nonlinear nature of the iteration makes analysis very difficult without linearization. In the $N=2$ case, the iteration can be analyzed directly; a stronger result is thus obtained. Similar stronger results may be true for larger N, but the analysis seems difficult.

Theorem 2: In the case $N=2$, the result of Theorem 1 can be strengthened to an "almost global" exponential stability result, *i.e.*, from any initial condition except a maximum, the iteration will converge exponentially to the desired minimum of J_1.

Proof: Without loss of generality, assume $X=\mathrm{diag}([\lambda_1\ \lambda_2])$. First notice that when $N=2$, $K=1$, so the set of transforms under consideration are described by a single scalar parameter. Dropping all unnecessary subscripts, (7) reduces to

$$A=VXU^T+UXV^T=\begin{bmatrix}(\lambda_2-\lambda_1)\sin 2\theta & (\lambda_2-\lambda_1)\cos 2\theta\\ (\lambda_2-\lambda_1)\cos 2\theta & -(\lambda_2-\lambda_1)\sin 2\theta\end{bmatrix}$$

and

$$Y=TXT^T=\begin{bmatrix}\lambda_1\cos^2\theta+\lambda_2\sin^2\theta & \frac{1}{2}(\lambda_2-\lambda_1)\sin 2\theta\\ \frac{1}{2}(\lambda_2-\lambda_1)\sin 2\theta & \lambda_1\sin^2\theta+\lambda_2\cos^2\theta\end{bmatrix}.$$

Simplifying (8) gives

$$\frac{\partial J_1}{\partial\theta}=2(Y_{12}A_{12}+Y_{21}A_{21})=(\lambda_2-\lambda_1)^2\sin 4\theta.$$

Thus the iteration to analyze is

$$\theta_{k+1}=\theta_k-\alpha(\lambda_2-\lambda_1)^2\sin 4\theta_k. \tag{11}$$

We immediately see that all multiples of $\pi/4$ are fixed points; the even multiples correspond to the desired transforms and the odd multiples are the only initial conditions for which the the iteration does not converge to a diagonalizing transform. For convenience, we consider only $0 < |\theta_0| < \pi/4$; other cases are similar. We will show that $\lim_{k\to\infty} \theta_k = 0$. Suppose $0 < \theta_0 < \pi/4$. Then using $\sin 4\theta \leq 4\theta$ and $\alpha < (\lambda_2 - \lambda_1)^{-2}/2$ one can show that $\alpha(\lambda_2 - \lambda_1)^2 \sin 4\theta_0 \in (0, 2\theta_0)$. Thus $|\theta_1| < |\theta_0|$. The $-\pi/4 < \theta_0 < 0$ case is similar. Since (11) is a strictly contractive mapping on $(-\pi/4, \pi/4)$ the iteration must converge to the only fixed point in the interval, zero. ■

C.2 Minimization of J_2

We will continue to use the notation introduced in Section IV-C.2. ∇J_2 is given elementwise by

$$\begin{aligned} \frac{\partial J_2}{\partial \theta_k} &= \frac{\partial}{\partial \theta_k} \prod_{i=1}^{N} Y_{ii} = \sum_{m=1}^{N} \left(\frac{\partial Y_{mm}}{\partial \theta_k} \prod_{i=1,\, i\neq m}^{N} Y_{ii} \right) = \sum_{m=1}^{N} \left(A_{mm}^{(k)} \prod_{i=1,\, i\neq m}^{N} Y_{ii} \right) \qquad (12) \\ &= J_2(T) \sum_{m=1}^{N} \left(\frac{1}{Y_{mm}} A_{mm}^{(k)} \right), \end{aligned}$$

where $A^{(k)}$ was defined in (7). As before, the gradient descent update is specified by (5).

Theorem 3: Denote the eigenvalues of X by $\lambda_1, \lambda_2, \ldots, \lambda_N$ and let $\Theta_\star$ correspond to a diagonalizing transform for X. Then for Θ_0 sufficiently close to $\Theta_\star$ the gradient descent algorithm described by (5) and (12) converges to $\Theta_\star$ if

$$0 < \alpha < \left[J_{\min} \max_{i,j} \frac{(\lambda_i - \lambda_j)^2}{\lambda_i \lambda_j} \right]^{-1}$$

where $J_{\min} = \prod_{i=1}^{N} \lambda_i$.

Proof: The method of proof is again to linearize the autonomous, nonlinear, discrete-time dynamical system that we have implicitly defined, and again the analysis is simplified by assuming that $X = \mathrm{diag}([\lambda_1\ \lambda_2\ \cdots \lambda_{\mathrm{N}}])$.

Differentiating (12) gives

$$\begin{aligned} \frac{\partial^2 J_2}{\partial \theta_\ell\, \partial \theta_k} &= \frac{\partial}{\partial \theta_\ell} \sum_{m=1}^{N} \left(A_{mm}^{(k)} \prod_{i=1,\, i\neq m}^{N} Y_{ii} \right) \\ &= \sum_{m=1}^{N} \left[\left(\frac{\partial A_{mm}^{(k)}}{\partial \theta_\ell} \prod_{i=1,\, i\neq m}^{N} Y_{ii} \right) + \left(A_{mm}^{(k)} \frac{\partial}{\partial \theta_\ell} \prod_{i=1,\, i\neq m}^{N} Y_{ii} \right) \right]. \qquad (13) \end{aligned}$$

When (13) is evaluated at $\Theta = \mathbf{0}$, the second term does not contribute because the diagonal of $A^{(k)}$ is zero for all k. Evaluation of $\frac{\partial A^{(k)}_{mm}}{\partial \theta_\ell}$ is somewhat tedious and is left for Appendix D. The result is summarized as

$$\frac{\partial A^{(k)}_{mm}}{\partial \theta_\ell} = \begin{cases} 2(\lambda_{j_k} - \lambda_{i_k}) & \text{if } k = \ell \text{ and } m = i_k; \\ 2(\lambda_{i_k} - \lambda_{j_k}) & \text{if } k = \ell \text{ and } m = j_k; \\ 0 & \text{otherwise;} \end{cases} \tag{14}$$

where (i_k, j_k) is related to k as before. Combining (13) and (14) gives

$$F_{k\ell} = \left[\frac{\partial^2 J_2}{\partial\theta_\ell\,\partial\theta_k}\right]_{\Theta=\mathbf{0}} = \begin{cases} \frac{2\,J_{\min}(\lambda_{i_k}-\lambda_{j_k})^2}{\lambda_{i_k}\lambda_{j_k}} & \text{if } k = \ell, \\ 0 & \text{otherwise.} \end{cases}$$

Requiring the eigenvalues of $I - \alpha F$ to lie in the unit circle completes the proof. ■

In the $N = 2$ case (but *not* in general) the two gradient descent algorithms that we have derived are equivalent. Hence Theorem 2 applies to the descent with respect to J_2 also. Again, we expect that the convergence result of Theorem 3 can be strengthened for general N, but the analysis seems difficult.

C.3 Comparison of descent methods

The linearizations used in the proofs of Theorems 1 and 3 facilitate easy analysis of the rates of convergence of the two descent methods. Consider the descent with respect to J_1. Using (10) we can approximate the error in the kth component of Θ at the nth iteration by $c[1-4\alpha(\lambda_{i_k}-\lambda_{j_k})^2]^n$. If we assume for the moment that we know $(\lambda_{i_k}-\lambda_{j_k})^2$, we could choose α to make the bracketed quantity equal to zero; then modulo the linearization, the convergence is in one step. The problem is that even if we could do this, the other components of Θ might not converge quickly or converge at all. Thus a quantity of fundamental interest in using the descent with respect to J_1 is the variability of $(\lambda_{i_k} - \lambda_{j_k})^2$, which we will call the *pseudo-eigenvalue spread* (since it is analogous to the eigenvalue spread in LMS adaptive filtering [11]):

$$s_1(X) = \frac{\max_{i,j}(\lambda_i - \lambda_j)^2}{\min_{i,j}(\lambda_i - \lambda_j)^2}$$

The corresponding quantity for the descent with respect to J_2 is

$$s_2(X) = \frac{\max_{i,j} \frac{(\lambda_i-\lambda_j)^2}{\lambda_i\lambda_j}}{\min_{i,j} \frac{(\lambda_i-\lambda_j)^2}{\lambda_i\lambda_j}}.$$

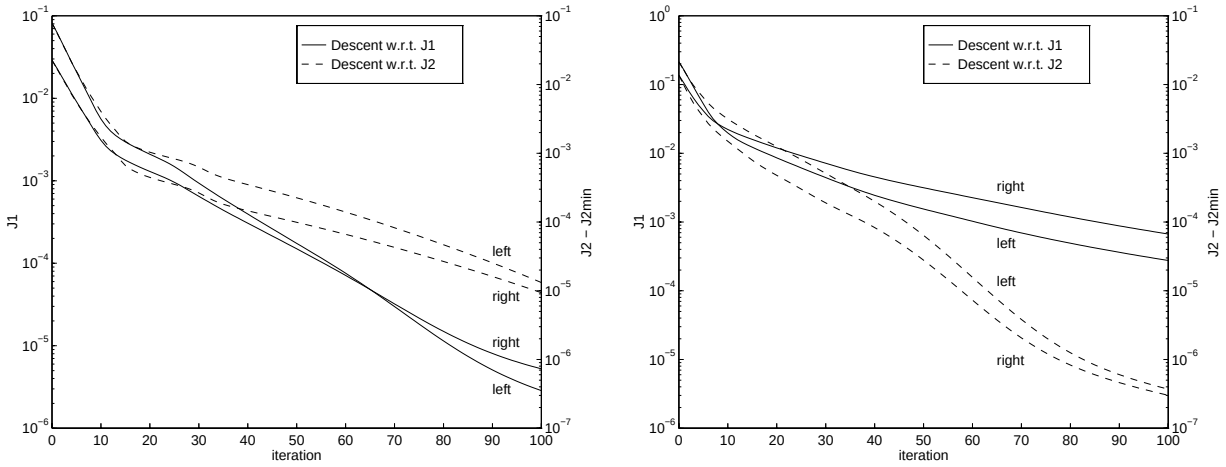


Fig. 2. Simulations of the gradient descent algorithms. In each case α is set to half the maximum value for stability and results are averaged over 100 randomly chosen initial conditions Θ_0. The relative performances of descents with respect to J_1 and J_2 is as predicted by the pseudo-eigenvalue spread. Parts (a) and (b) are for matrices X_1 and X_2, respectively. (The curve labels refer to the left and right y-axes.)

The difference between $s_1(X)$ and $s_2(X)$ suggests that the superior algorithm will depend on X along with the choice of α. This is confirmed through the following calculations and simulations. Consider the matrices $X_1 = \mathrm{diag}([1,\ \frac{7}{8},\ \frac{5}{8},\ \frac{1}{2}])$ and $X_2 = \mathrm{diag}([1,\ \frac{1}{2},\ \frac{1}{4},\ \frac{1}{8}])$, for which we have $s_1(X_1) = 16 < 28 = s_2(X_1)$ and $s_1(X_2) = 49 > \frac{49}{4} = s_2(X_2)$. Based on the pseudo-eigenvalue spreads we expect a descent with respect to J_1 to perform better than a descent with respect to J_2 for diagonalizing X_1, and vice versa for X_2. Simulations were performed with α at half the maximum value for stability and 100 randomly selected initial conditions Θ_0. The averaged results, shown in Figure 2, indicate that the performance is as predicted.

D. Nonparametric Methods

As we have noted, finding the optimal transform is equivalent to finding an eigendecomposition of a symmetric matrix. The best algorithms (rated in terms of the number of floating point operations) for the symmetric eigenproblem do not use a parameterization of a diagonalizing transform as we have done in the preceding sections. The best algorithms to date for computing the eigendecomposition of a single symmetric matrix are variations of the QR algorithm. However, these algorithms do not allow one to take advantage of knowledge of approximate eigenvectors, as one

would have with a slowly-varying sequence of X matrices. In this section we briefly introduce Jacobi methods, which allow this prior information to be effectively incorporated. Details on QR and Jacobi algorithms can be found in [3, §8.5].

The idea of the classical Jacobi algorithm is to at each iteration choose a Givens rotation to reduce the off-diagonal energy as much as possible. More specifically, the algorithm produces a sequence $\{T_k\}$ and also keeps track of $A_k = T_k X T_k^T$. If the Givens rotation $\widetilde{G}_{i,j,\theta}$ (see (3)) is chosen in computing T_{k+1}, the maximum reduction in the off-diagonal energy (by correctly choosing θ) is $(A_k)_{ij}^2$, thus the best choice for (i,j) is that which maximizes $(A_k)_{ij}^2$. It is a greedy minimization of J_1, but since Givens rotations do not commute, it is hard to interpret it in terms of the parameterization we used earlier.

A drawback of the classical Jacobi algorithm is that while each iteration requires only $O(N)$ operations for the updates to T_k and A_k, choosing (i,j) requires $O(N^2)$ operations. This can be remedied by eliminating the search step and instead choosing (i,j) in a predetermined manner. This is called the cyclic Jacobi algorithm and each cycle through the $K = N(N-1)/2$ distinct (i,j) pairs is called a *sweep*.

To provide a basis of comparison with the results of Sections IV-B and IV-C, simulations of the cyclic Jacobi algorithm were performed with on $X = \mathrm{diag}([1,\ \frac{1}{2},\ \frac{1}{4},\ \frac{1}{8}])$ with random initial transforms corresponding to the random initial parameter vectors used before. The averaged results of 400 simulations are shown in Figure 3. Note that the x-axis shows the number of rotations, not the number of sweeps.

An attractive feature of the cyclic Jacobi algorithm is that the updates can be partitioned into set of "noninteracting" rotations, *i.e.*, rotations involving disjoint sets of rows and columns. These noninteracting rotations can be done in parallel. All Jacobi algorithms have the advantage that a good initial transform speeds convergence.

E. Comments and Comparisons

Comments on the relative merits of random search, gradient descent, and Jacobi methods are in order. By comparing Figures 1–3, it is clear that the cyclic Jacobi method gives the fastest convergence rate in terms of the number of iterations or rotations. Since the Jacobi method also has the lowest complexity, for general purpose eigendecomposition the remaining methods seem to be only of academic interest.

A potential benefit of the random search and gradient descent methods is that they operate

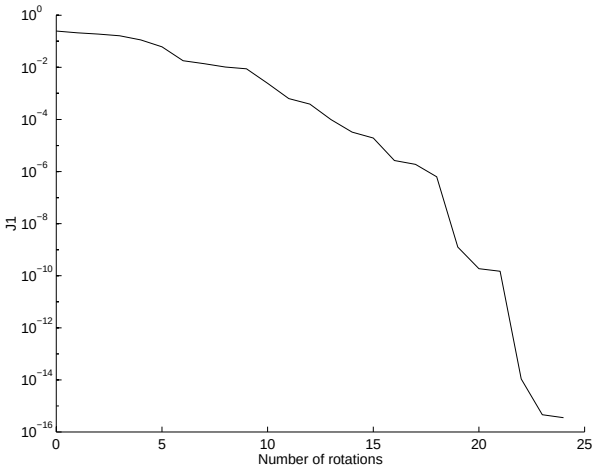


Fig. 3. Simulations of the cyclic Jacobi algorithm on $X = \mathrm{diag}([1,\ \frac{1}{2},\ \frac{1}{4},\ \frac{1}{8}])$ with randomly chosen initial transform T_0.

directly on a minimal parameterization of the transform matrices of interest. Of course, a transform matrix could be determined using a Jacobi method and then parameterized afterward.

V. Adaptive Transform Coding Update Methods

In the previous section we established a set of algorithms for iteratively determining the optimal transform assuming that the source correlation matrix X_n is constant. Recalling our overall strategy, we would now like to apply these algorithms in an adaptive setting.

The traditional implementation approach would be to calculate a sequence of estimates $\{\hat{X}_n\}$ using a windowed time average and to use these averages in the adaptive algorithms. The extreme case of this approach is to use a time average over only one sample, *i.e.*, $\hat{X}_n = x_n x_n^T$. This results in a computational savings and—in the case of gradient descent parameter search—gives an algorithm very much in the spirit of LMS. Specifically in a transform coding application, it may be desirable to eliminate the need for side information by putting quantization inside the adaptation loop. These implementation possibilities are described in detail in the the remainder of this section.

In the interest of brevity, simulation results are not provided for each combination of cost function, implementation structure, and search algorithm. The greatest emphasis is placed on stochastic gradient parameter surface search.

A. Explicit Autocorrelation Estimation

The most obvious way to implement an adaptive transform coding system is to use a windowed correlation estimate of the form

$$\hat{X}_n = \frac{1}{M} \sum_{k=n-M+1}^{n} x_k x_k^T. \tag{15}$$

If the true correlation is constant, then $\hat{X}_n$ is elementwise an unbiased, consistent estimator of X [11]. There will be "estimation noise" (variance in $\hat{X}_n$ due to having finite sample size) which decreases monotonically with M. If $\{X_n\}$ is slowly varying, there will also be "tracking noise" (mismatch between $\hat{X}_n$ and X_n caused by the causal observation window) which increases with M. Thus in the time-varying case there is a tradeoff, controlled by M, and one can expect there to be an optimal value of M depending on the rate at which $\{X_n\}$ varies.

To illustrate the ability to track a time-varying source and the dependence on M, we construct the following synthetic source: For each time $n \in \mathbb{Z}^+$, x_n is a zero-mean jointly Gaussian vector with correlation matrix

$$X_n = U_n^T \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{4} \end{bmatrix} \cdot U_n,$$

where U_n is a time-varying unitary matrix specified as

$$U_n = G_{1,\omega_1 n+\varphi_1} G_{2,\omega_2 n+\varphi_2} G_{3,\omega_3 n+\varphi_3}.$$

U_n is an ideal transform to be used at time n. The ω_i's are fixed "angular velocities" to be tracked and the φ_i's are independent, uniformly distributed phases. Averaging over randomly selected phases removes any periodic components from simulation results.

This source was used in simulations of the linear search with respect to J_1. For all the simulations $\alpha = 3$ and $\sigma = 0.01$. In the first set of experiments (see Figure 4(a)) $\omega_1 = \omega_2 = \omega_3 = 0$. Since the source is not time varying, there is no tracking noise and the estimation noise decreases as M is increased, so the overall performance improves as M in increased. The second and third sets of experiments use $\omega_1 = \omega_2 = \omega_3 = 0.001$ and $\omega_1 = \omega_2 = \omega_3 = 0.002$, respectively. Now since the source is time varying, the performance does not improve monotonically as M is increased because as the estimation noise decreases, the tracking noise increases. For the slower varying source (see Figure 4(b)) the performance improves as M is increased from 5 to 20 and

then is about the same for $M = 40$. For the faster varying source (see Figure 4(c)) the tracking noise is more significant so the best value of M is lower. A faster varying source may also justify a larger value of α.

The estimate (15) implicitly uses a rectangular window to window the incoming data stream, so each sample vector is equally weighted. One way to more heavily weight the later sample vectors is to use a "forgetting factor" β (as in Recursive Least Squares [12]), which is equivalent to using an exponential window:

$$\hat{X}_n = \beta \hat{X}_{n-1} + (1-\beta) x_n x_n^T.$$

This scheme also reduces memory requirements.

B. Stochastic Update

If we take the autocorrelation estimation of the previous section to its extreme of estimating the autocorrelation based on a single sample vector, we get

$$\hat{X}_n = x_n x_n^T. \tag{16}$$

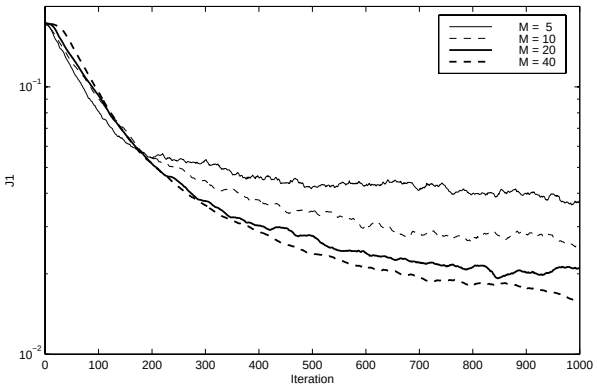
The use of this extremely simple estimate simplifies the calculations associated with parameter surface search. We refer to this as a *stochastic implementation* because it is the result of replacing an expected value by its immediate, stochastic value.

Both random search methods require calculation of J. For a general $X \in \mathbb{R}^{N \times N}$, computing $TXT^T = G_1 G_2 \cdots G_K X G_K^T \cdots G_2^T G_1^T$ requires $8KN$ multiplications and $4KN$ additions because each multiplication by a Givens matrix requires $4N$ multiplications and $2N$ additions. With the rank-one $\hat{X}_n$ given by (16), we can first write

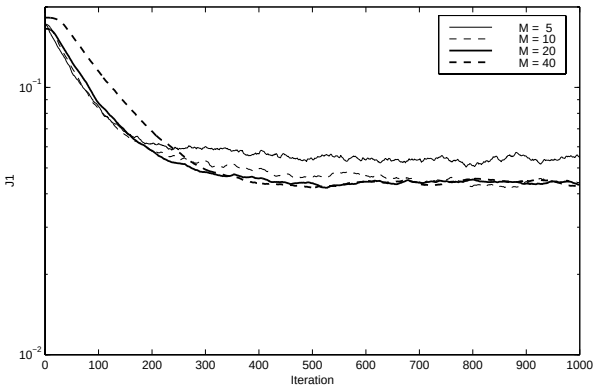
$$\begin{aligned} T\hat{X}_n T^T &= G_1 G_2 \cdots G_K x_n x_n^T G_K^T \cdots G_2^T G_1^T \\ &= (G_1 G_2 \cdots G_K x_n)(G_1 G_2 \cdots G_K x_n)^T. \end{aligned}$$

Then since multiplying a vector by a Givens matrix requires 4 multiplications and 2 additions, the bracketed terms can be computed with $4K$ multiplications and $2K$ additions. Now $J_1(T)$ can be computed with K additional multiplications and $K-1$ additional additions or $J_2(T)$ can be computed with N additional multiplications. The computation of ∇J is similarly simplified.

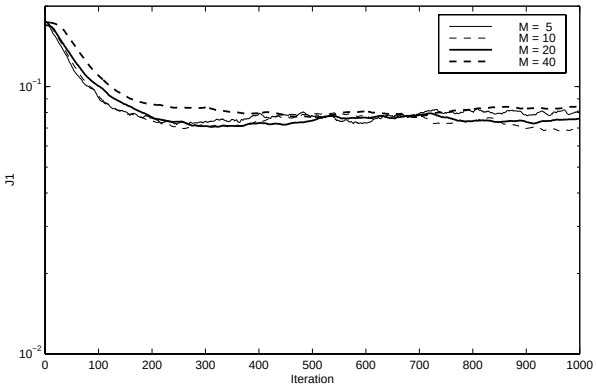
We have simulated the stochastic implementation of gradient descent parameter search for the source described in the previous section (see Figure 5). There is a single parameter to choose:



(a) $\omega_1 = \omega_2 = \omega_3 = 0$



(b) $\omega_1 = \omega_2 = \omega_3 = 0.001$



(c) $\omega_1 = \omega_2 = \omega_3 = 0.002$

Fig. 4. Simulations of linear search with respect to J_1 with explicit correlation estimation. The source is slowly varying as described in the text. M is the length of the data window. Fixed parameters: $\alpha = 3$, $\sigma = 0.01$. Results are averaged over 400 randomly chosen initial conditions Θ_0 and source phases φ_1, φ_2, φ_3.

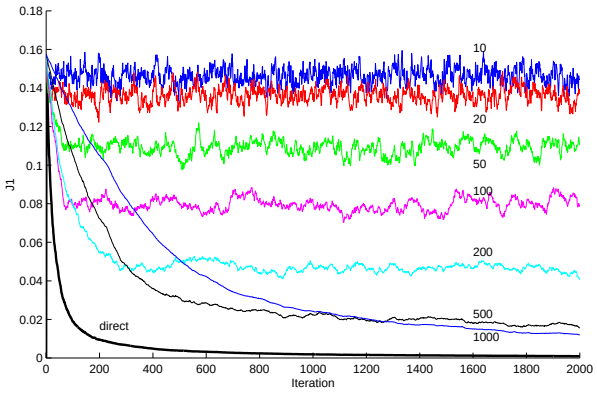
the step size α. Using Theorems 1 and 3 gives maximum step sizes of $\frac{8}{9}$ and $\frac{32}{9}$ for descent with respect to J_1 and J_2, respectively. These theorems apply only to iterative computations with *exact* knowledge of the correlation matrix; however, they provide rough guidelines for step size choice in the stochastic setting.

When the source distribution is time-invariant ($\omega_1 = \omega_2 = \omega_3 = 0$ for the source we are considering), the effect of the step size α is easy to discern. A larger step size reduces the adaptation time constants, so steady-state performance is reached more quickly. However, because the parameter vector Θ is adapted based on each source vector, the steady-state performance has a "noisy" stochastic component. This "excess" in J increases as the step size is increased. We have not attempted to characterize this analytically. Qualitatively it is similar to the "excess" mean-square error in LMS filtering [2]. Referring to Figure 5(a), the steady-state value of J_1 decreases monotonically as α is decreased, but the convergence is slower. Because the source is time-invariant, there is a conceptually simple alternative to the stochastic gradient descent which provides a bound to attainable performance. This is to use all the source vectors observed thus far to estimate the correlation, using (15) with $M = n$, and computing the eigendecomposition of the correlation estimate to full machine precision. This bound is the lowest curve in Figure 5(a).

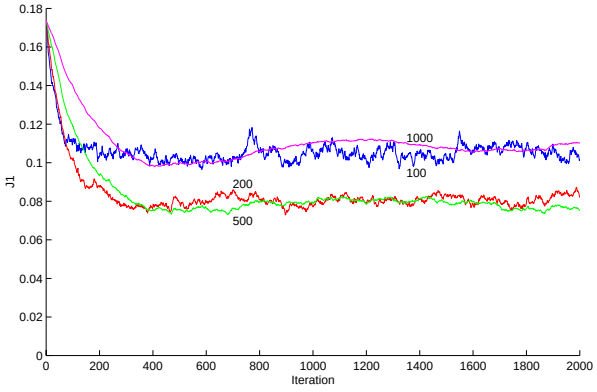
The situation is more complicated when the source distribution is time-varying. Now the step size effects the ability to track the time variation along with determining the steady-state noise and speed of convergence. Figures 5(b) and (c) show the results of simulations with $\omega_1 = \omega_2 = \omega_3 = 0.001$ and $\omega_1 = \omega_2 = \omega_3 = 0.002$, respectively. In the first of these simulations, the best performance is achieved for α between $\frac{8}{9}/500$ and $\frac{8}{9}/200$. The larger of these gives slightly faster convergence and the small gives slightly lower steady-state error. For the faster-varying source, $\frac{8}{9}/500$ is too small for effectively tracking the source.

C. Quantized Stochastic Implementation

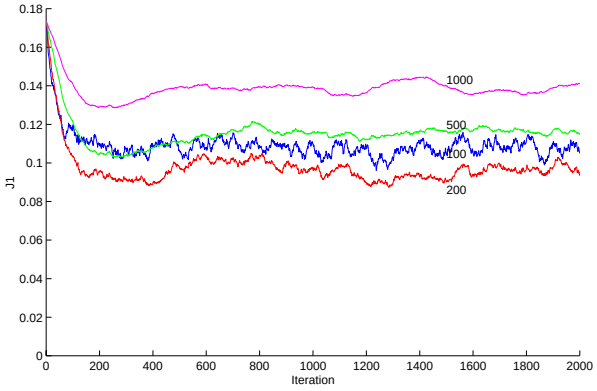
In adaptive transform coding, if the transform adaptation is based upon the incoming *uncoded* data stream, then in order for the decoder to track the encoder state, the transform adaptation must be described over a side information channel. This situation, which is commonly called *forward-adaptive*, is depicted in Figure 6(a). The need for side information can be eliminated if the adaptation is based on the coded data, as shown in Figure 6(b). This *backward-adaptive* configuration again has an analogy in adaptive FIR Wiener filtering: In adaptive linear predictive coding, where the linear predictor is in fact an adaptive FIR Wiener filter, making the adaptation



(a) $\omega_1 = \omega_2 = \omega_3 = 0$

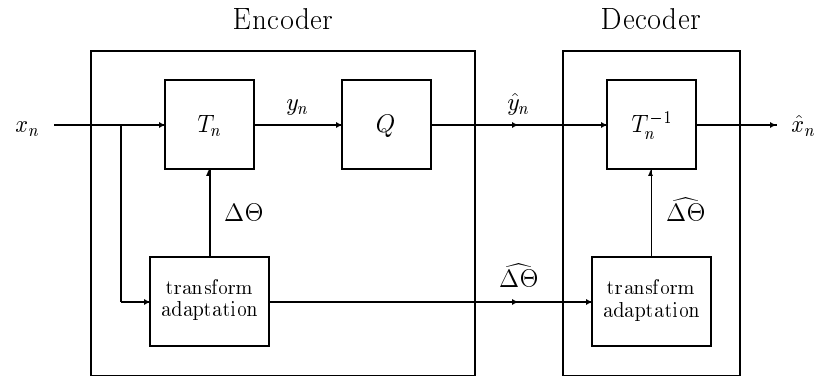


(b) $\omega_1 = \omega_2 = \omega_3 = 0.001$

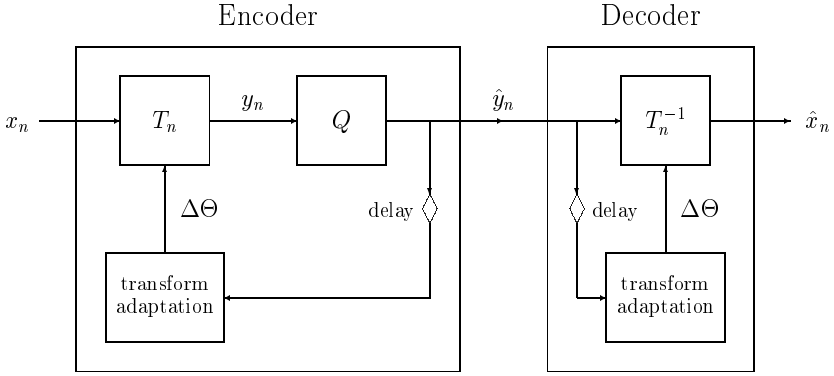


(c) $\omega_1 = \omega_2 = \omega_3 = 0.002$

Fig. 5. Simulations of stochastic gradient descent with respect to J_1. The source is slowly varying as described in the text. Step sizes are given by $\alpha = \alpha_{\max}/\gamma$, where $\alpha_{\max} = \frac{8}{9}$ is the maximum step size for stability predicted by Theorem 1. The curves are labeled by the value of γ. Results are averaged over 400 randomly chosen initial conditions Θ_0 and source phases φ_1, φ_2, φ_3. In (a) the performance is also compared to computing an exact eigendecomposition of a correlation estimate based on all the sample vectors observed thus far.



(a) Forward-adaptive transform coding system



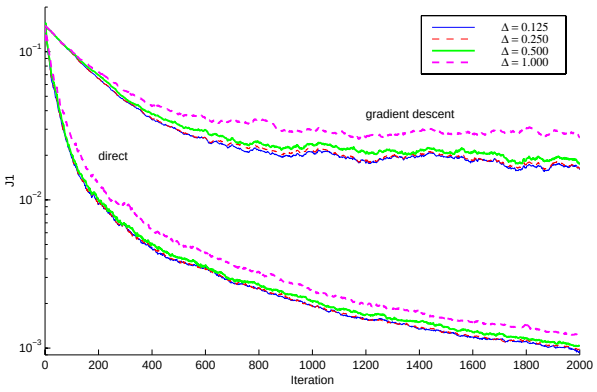
(b) Backward-adaptive transform coding system

Fig. 6. Structural comparison between forward- and backward-adaptive systems. The backward-adaptive system does not require a side information channel to convey transform state.

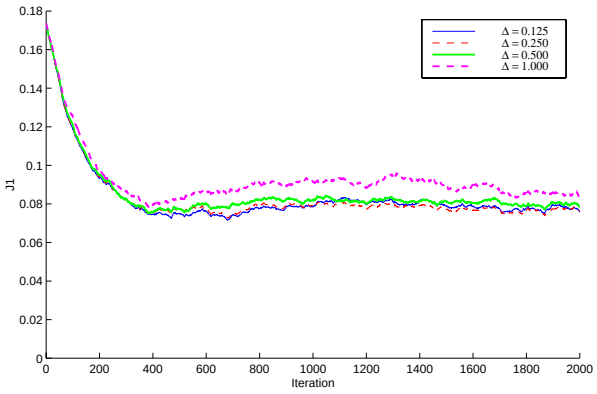
depend on quantized data yields adaptive differential pulse code modulation (ADPCM).[3]

We have simulated the stochastic gradient descent in the backward-adaptive configuration. Since quantization is an irreversible reduction in information, it must be at least as hard to estimate the moments of a signal from a quantized version as it is from the original unquantized signal. Thus we expect the convergence rate to be somewhat worse in the backward-adaptive configuration. Figure 7(a) shows simulation results for a time-invariant source ($\omega_1 = \omega_2 = \omega_3 = 0$). The lower set of curves is for direct computation as in Figure 5(a) and the upper set of curves is for stochastic gradient descent with step size $\alpha = \frac{8}{9}/500$. With quantization step size $\Delta = 0.125$ or 0.25, the rate of convergence is almost indistinguishable from the unquantized case.

[3]Note that ADPCM is often used to refer to a system with adaptive quantization. However, quantization adaptation is beyond the scope of this paper.



(a) Time-invariant source ($\omega_1 = \omega_2 = \omega_3 = 0$)



(b) Slowly-varying source ($\omega_1 = \omega_2 = \omega_3 = 0.001$)

Fig. 7. Simulations of stochastic gradient descent with respect to J_1. in backward-adaptive configuration. Step sizes are given by $\alpha = \alpha_{\max}/500$, where $\alpha_{\max} = \frac{8}{9}$ is the maximum step size for stability predicted by Theorem 1. The curves are labeled by the value of the quantization step size Δ. Results are averaged over 400 randomly chosen initial conditions Θ_0 and source phases φ_1, φ_2, φ_3. In (a) the performance is also compared to computing an exact eigendecomposition of a correlation estimate based on all the (quantized) sample vectors observed thus far.

As the quantization becomes coarser, the convergence slows. Notice that with direct computation, quantization does not seem to lead to a nonzero steady-state error. This is suggestive of universal performance of the backward-adaptive scheme [13]; further discussion of this is beyond the scope of this paper.

For a slowly varying source ($\omega_1 = \omega_2 = \omega_3 = 0.001$; see Figure 7(b)), we again have that the performance with $\Delta = 0.125$ or 0.25 is indistinguishable from the performance without quantization. The convergence slows as the quantization becomes coarser, but here there may

also be a small increase in steady-state error.

D. Specialization for a Scalar Source

In many applications with processing of vectors, the vectors are actually generated by forming blocks a scalar-valued source. The methods developed in this paper are general and hence applicable to this case. However, a few specific refinements facilitate performance better than in the general case.

Suppose the original scalar source is a wide-sense stationary process $\{z_n\}$ which we observe for $n \geq 1$ and we generate a vector source $\{x_n\}$ by forming blocks of length N. Then the correlation matrix $X = E[x_n x_n^T]$ is a symmetric, Toeplitz matrix with $X_{ij} = r_z(i-j) = E[z_i z_j]$.

One consequence of the symmetric, Toeplitz structure of X is that there are actually less than $K = N(N-1)/2$ independent parameters to estimate to find a diagonalizing transform. For $N = 3$, for example, one can show that

$$X = \begin{bmatrix} a & b & c \\ b & a & b \\ c & b & a \end{bmatrix}$$

has always as an eigenvector $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^T$ and that it suffices to consider transforms of the form

$$T = \begin{bmatrix} -\sqrt{2}/2 & 0 & \sqrt{2}/2 \\ \zeta & \sqrt{1-2\zeta^2} & \zeta \\ \sqrt{1-2\zeta^2}/\sqrt{2} & -\sqrt{2}\zeta & \sqrt{1-2\zeta^2}/\sqrt{2} \end{bmatrix}.$$

This can be used to derive new performance surface search methods with fewer parameters.

A second consequence is that estimates better than (15) can be used. Having observed M N-tuples from the source, (15) gives

$$\hat{X}_{ij} = \frac{1}{M}\sum_{n=1}^{M}(x_n)_i(x_n)_j = \frac{1}{M}\sum_{n=1}^{M} z_{N(n-1)+i} z_{N(n-1)+j}. \tag{17}$$

Each of the terms of (17) has expected value $r_z(i-j)$ and by averaging over M observations we clearly get an unbiased, consistent estimate. However, with MN samples we can actually average over $MN-(i-j)$ terms to get a much lower variance estimate:

$$\hat{X}_{ij} = \widehat{r_z}(i-j) = \frac{1}{MN-(i-j)} \sum_{n=1}^{MN-(i-j)} z_n z_{n+(i-j)}.$$

For a time-varying source, either a finite window or a forgetting factor could be used.

VI. Conclusions

This paper has introduced a new class of algorithms for computing the eigenvectors of a symmetric matrix. These algorithms are potentially useful for adaptive transform coding or online principal component analysis. The development is conceptually summarized as follows: A matrix of eigenvectors forms an orthogonal diagonalizing similarity transformation; it suffices to consider orthogonal matrices which are parameterized as a product of Givens rotations; and appropriate parameter values can be found as an unconstrained minimization.

The key is the formulation of unconstrained minimization problems over a minimal number of parameters. Borrowing from the adaptive filtering literature, we have applied linear and fixed step random search and gradient descent to the resulting minimization problems. In the gradient descent case we derived step size bounds to ensure convergence in the absence of estimation noise. Through simulations we demonstrated that in the presence of estimation noise, the gradient descent converges when the step size is chosen small relative to the bound.

In a transform coding application, one may want to use a backward-adaptive configuration in which the adaptation is driven by quantized data so that the decoder and encoder can remain synchronized without the need for side information. As long as the quantization is not too coarse, the algorithms presented here seem to converge.

Appendices

I. Brief Review of Transform Coding

The fundamental purpose of source coding is to remove redundancy. One elementary form of redundancy is correlation between components of a vector. Intuitively, transform coding (and the choice of the transform therein) is based on removing this simple form of redundancy.

Let $\{x_n\}_{n\in\mathbb{Z}^+}$ be a sequence of $\mathbb{R}^N$-valued random vectors.[4] A complete transform coding system for this source is shown in Figure 8. Applying the orthogonal transform $T_n \in \mathbb{R}^{N\times N}$ to x_n gives y_n, which is quantized with the quantization function Q_n to give $\hat{y}_n$. The encoding is completed by applying an entropy code E_n to $\hat{y}_n$. The entropy coder may operate with memory and/or delay in order to improve its performance. The corresponding inverse operations are

[4] All quantities are real throughout the paper; some results could easily be extended to the complex case. Depending on the context, the time index n may be suppressed and a subscript may be used to distinguish between components of a vector.

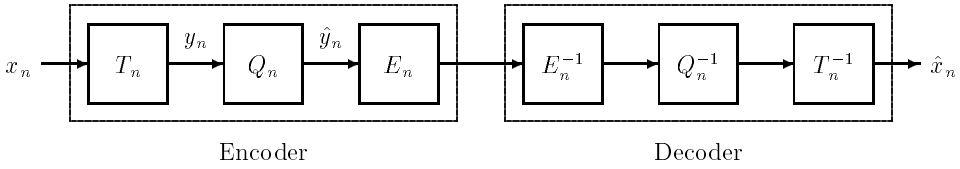


Fig. 8. Basic transform coding system.

performed at the decoder.

In classical transform coding theory, as introduced in [14] and analyzed in detail in [15], the source vectors are assumed to have identical distributions and are treated independently.[5] The problem is to find the orthogonal[6] transform T and quantizer Q such that for a fixed bit rate R (bits per scalar coefficient) the distortion $D = E\|x_n - \hat{x}_n\|^2$ is minimized.

Denote the autocorrelation of x by $X = E[xx^T]$.[7] The autocorrelation of y is given by $Y = E[yy^T] = E[Txx^TT^T] = TXT^T$. We arrive at the optimality of the Karhunen-Loève Transform (KLT) as follows (see [16] for more details). Because of the orthogonality of T, $\|x_n - \hat{x}_n\|^2 = \|y_n - \hat{y}_n\|^2$, so the distortion in x is exactly the distortion in quantizing y. Assuming optimal scalar quantization and that the high rate approximation holds, the distortion in quantizing the coefficient y_i is $D_i = h_i Y_{ii}^2 2^{-2R_i}$, where R_i is the bit rate for that component and the constant h_i depends on the p.d.f. $f_i(t)$ of the normalized random variable y_i/Y_{ii}:

$$h_i = \frac{1}{12}\left(\int_{-\infty}^{\infty} [f_i(t)]^{1/3}\, dt\right)^3$$

Then assuming that the h_i's are equal,[8] $h_i = h$, $i = 1, 2, \ldots, N$, the optimal bit allocation among the transform coefficients results in an overall distortion of

$$D = Nh\rho^2 2^{-2R}, \tag{18}$$

where

$$\rho^2 = \left(\prod_{i=0}^{N} Y_{ii}^2\right)^{1/N}. \tag{19}$$

[5]Transform coding is often introduced in the case where x_n is N consecutive samples of a stationary (scalar) source. This is one case in which the x_n's are identically distributed.

[6]The reasons for constraining the transform T to be orthogonal are somewhat subtle. It is often said that otherwise the inverse transform will enhance quantization errors, but detailed justification is difficult. For a Gaussian signal, it was shown in [15] that T should be orthogonal and that the decoder should use T^{-1}.

[7]We have used X in place of the usual R_x in order to reduce the need for multiple subscripts.

[8]This is the case when the source is Gaussian.

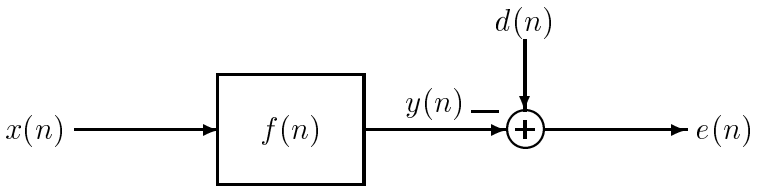


Fig. 9. Canonical configuration for Wiener filtering. The objective is to design the filter $f(n)$ such that the power of $e(n)$ is minimized.

An orthogonal T that minimizes (19) (and hence (18)) is one that diagonalizes X; *i.e.*, $Y = TXT^T$ is diagonal. (A simple proof based on the arithmetic/geometric mean inequality is given in [16].) Among such transforms, one that leaves the diagonal of Y sorted in nonincreasing order is called a Karhunen-Loève Transform (KLT) for the source. The KLT is unique (up to a choice of signs for each row) if the eigenvalues of X are distinct.

The optimality of the KLT as established above only applies to high rate coding of Gaussian sources. However, the KLT is believed to be a good transform for transform coding in other situations as well because the principle of decorrelating as a means of reducing redundancy is generally applicable. The reader is referred to [16] for more details on transform coding.

II. Brief Review of Adaptive FIR Wiener Filtering

The canonical Wiener filtering[9] problem is described as follows. Let $x(n)$ and $d(n)$ be jointly wide-sense stationary, zero-mean, scalar random processes. Design a linear filter $f(n)$ such that the mean-squared error between the desired signal $d(n)$ and the output of the filter $y(n) = x(n) * f(n)$ is minimized (see Figure 9). Two common applications are separating signal from noise and channel equalization. For denoising, $x(n) = d(n) + w(n)$ where $w(n)$ is unknown, but has known spectral density and is uncorrelated with $d(n)$. For equalization, $x(n) = d(n) * c(n)$, where $c(n)$ is a channel impulse response.

We consider here the case where $f(n)$ is constrained to be a causal, L-tap FIR filter. This and other cases are discussed in detail in [11].

Finding the optimal filter is conceptually simple once we select a convenient vector notation. Let $\bar{f} = [f(0),\, f(1),\, \ldots,\, f(L-1)]^T$ and $\bar{x}_n = [x(n),\, x(n-1),\, \ldots,\, x(n-L+1)]^T$. Then $e(n) = d(n) - \bar{x}_n^T \bar{f}$. The power of $e(n)$ is a quadratic function of the filter vector:

$$J(f) = E[e(n)^2]$$

[9]The anonymous designation of "optimal least-squares filtering" is also used.

It is this function which we call the *performance surface*, and finding the optimal filter is to find the parameter vector which yields the minimum of the performance surface.[10] It can be shown that the gradient of J with respect to $\bar{f}$ is given by

$$\nabla J = 2(X\bar{f} - \bar{r}_{dx}),$$

where (consistent with the previous section) $X = E[\bar{x}_n \bar{x}_n^T]$ and $\bar{r}_{dx} = E[\bar{x}_n d(n)]$. From this we can conclude that the optimal filter is described by

$$\bar{f}_{\text{opt}} = X^{-1}\bar{r}_{dx}. \tag{20}$$

There are two practical problems in applying the analytical solution (20). The first is that X and $\bar{r}_{dx}$ may be unknown and may depend on n. A remedy would be to estimate these moments as the incoming data is processed, giving $X(n)$ and $\bar{r}_{dx}(n)$. This leads to the second problem, which is that each update of the filter requires solving a linear system $X(n)\bar{f} = \bar{r}_{dx}(n)$.

The LMS or stochastic gradient algorithm addresses both of these problems. There are two main ideas. Firstly, instead of exactly minimizing J by using (20), iteratively update $\bar{f}$ by adding $-\alpha\nabla J$, where $\alpha > 0$ is called the step size. As long as α is chosen small enough, this procedure will converge to $\bar{f}_{\text{opt}}$; however, as long as we still require knowledge of X and $\bar{r}_{dx}$ this is not very useful. The second main idea is to replace X and $\bar{r}_{dx}$ by the simplest possible stochastic approximations: $X(n) \approx \bar{x}_n \bar{x}_n^T$ and $\bar{r}_{dx} \approx \bar{x}_n d(n)$. This yields the update equation for LMS:

$$\bar{f}(n+1) = \bar{f}(n) - 2\alpha(y(n) - d(n))\bar{x}_n \tag{21}$$

One normally studies the stability and rate of convergence of (21) by analyzing

$$\bar{f}(n+1) = \bar{f}(n) - 2\alpha(X\bar{f} - \bar{r}_{dx}).$$

This can be interpreted as ignoring the stochastic aspect of the algorithm or as looking at the mean of $\bar{f}$ and applying the so-called "independence assumption" [1].

III. Alternative Gradient Expressions

The gradient expressions given in Section IV-C were intended to facilitate Theorems 1 and 3. Alternative expressions for ∇J_1 and ∇J_2 are given in this appendix.

[10] Under some technical conditions, the minimum is unique.

We will use the chain rule to compute ∇J_ℓ, $\ell = 1, 2$, through

$$\frac{\partial J_\ell}{\partial \theta_k} = \sum_{i,j} \frac{\partial J_\ell}{\partial T_{ij}} \frac{\partial T_{ij}}{\partial \theta_k}.$$

Recalling the definitions of $U_{(a,b)}$ and $V_{k,\theta}$ from Section IV-C.1, if we define $B_{ij}^{(k)} = \partial T_{ij}/\partial \theta_k$, then differentiating (4) gives

$$B^{(k)} = U_{(1,k-1)} V_k U_{(k+1,K)}.$$

For both $\ell = 1$ and $\ell = 2$, the following intermediate calculation is useful:

$$\frac{\partial Y_{ab}}{\partial T_{ij}} = \frac{\partial}{\partial T_{ij}} \sum_{r=1}^{N} \sum_{s=1}^{N} T_{as} X_{sr} (T^T)_{rb} = \delta_{i-a} T_{b*} X_{*j} + \delta_{i-b} T_{a*} X_{*j},$$

where T_{b*} is the bth row of T and X_{*j} is the jth column of X.

Now

$$\frac{\partial J_1}{\partial T_{ij}} = \sum_{a \neq b} 2 Y_{ab} \frac{\partial Y_{ab}}{\partial T_{ij}} = \sum_{a \neq b} 2 Y_{ab} (\delta_{i-a} T_{b*} X_{*j} + \delta_{i-b} T_{a*} X_{*j}) = 4 e_i^T Y (I - e_i e_i^T) T X e_j,$$

where e_i is the column vector with one in the ith position and zeros elsewhere.

For J_2 we have

$$\frac{\partial J_2}{\partial T_{ij}} = \frac{\partial}{\partial T_{ij}} \prod_{\ell=1}^{N} Y_{\ell\ell} = \sum_{a=1}^{N} \frac{\partial Y_{aa}}{\partial T_{ij}} \prod_{\ell=1, \ell \neq a}^{N} Y_{\ell\ell} = \sum_{a=1}^{N} \frac{J}{Y_{aa}} 2 \delta_{i-a} T_{a*} X_{*j} = 2 \frac{J}{Y_{ii}} T_{i*} X_{*j},$$

IV. Evaluation of $\partial A_{mm}^{(k)}/\partial \theta_\ell$

In this appendix we derive (14). First consider the case $\ell = k$. Let $W_k = \frac{\partial}{\partial \theta} V_k$. Differentiating (7) gives

$$\begin{aligned} \frac{\partial}{\partial \theta_k} A^{(k)} &= U_{(1,k-1)} W_k U_{(k+1,K)} X U_{(1,K)}^T + U_{(1,k-1)} V_k U_{(k+1,K)} X U_{(k+1,K)}^T V_k^T U_{(1,k-1)}^T \\ &\quad + U_{(1,K)} X U_{(k+1,K)}^T W_k^T U_{(1,k-1)}^T + U_{(1,k-1)} V_k U_{(k+1,K)} X U_{(k+1,K)}^T V_k^T U_{(1,k-1)}^T, \end{aligned}$$

which upon evaluation at $\Theta = 0$ reduces to

$$\left. \frac{\partial}{\partial \theta_k} A^{(k)} \right|_{\Theta = \mathbf{0}} = W_k X + V_k X V_k^T + X W_k^T + V_k X V_k^T.$$

The simple structures of V_k and W_k allow one to now easily show that

$$\left.\frac{\partial}{\partial\theta_k}A^{(k)}\right|_{\Theta=\mathbf{0}} = \begin{bmatrix} 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & 2(\lambda_{j_k}-\lambda_{i_k}) & \cdots & 0 & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 2(\lambda_{i_k}-\lambda_{j_k}) & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \end{bmatrix} \begin{matrix} \\ \\ i_k \\ \\ j_k \\ \\ \\ \end{matrix} .$$

$$\begin{matrix} & & i_k & & j_k & & \end{matrix}$$

Now consider the case $\ell < k$. Differentiating (7) and evaluating at $\Theta = \mathbf{0}$ gives

$$\left.\frac{\partial}{\partial\theta_\ell}A^{(k)}\right|_{\Theta=\mathbf{0}} = V_\ell V_k X + V_k X V_\ell^T + X V_k^T V_\ell^T + V_\ell X V_k^T. \tag{22}$$

To satisfy (14) we would like to show that the diagonal of (22) is zero.

Lemma 2: For $\ell < k$ and $\Theta = 0$, the diagonal of $V_\ell V_k$ is zero.

Proof: Because $\ell < k$, we have either

(a) $i_\ell < i_k$; or

(b) $i_\ell = i_k$ and $j_\ell < j_k$.

Recall also that $i_\ell < j_\ell$ and $i_k < j_k$.

The only potentially nonzero elements of $V_\ell V_k$ are in the (i_ℓ, i_k), (i_ℓ, j_k), (j_ℓ, i_k), and (j_ℓ, j_k) positions. The (i_ℓ, j_k) element can not be on the diagonal because either $i_\ell < i_k < j_k$ or $i_\ell = i_k < j_k$; similarly for the (j_ℓ, i_k) element. The (i_ℓ, i_k) element is $-\delta(j_\ell - j_k)$ and hence when this element is on the diagonal, it is zero; similarly for the (j_ℓ, j_k) element. ■

Corollary 1: For $\ell < k$ and $\Theta = 0$, the diagonals of $V_k V_\ell^T$, $V_k^T V_\ell^T$, and $V_\ell V_k^T$ are zero.

Since X is diagonal, Lemma 2 and Corollary 1 can be combined to show that the diagonal of (22) is zero.

The $\ell > k$ case is similar to the $\ell < k$ case.

References

[1] B. Widrow, J. M. McCool, M. G. Larimore, and C. R. Johnson, Jr., "Stationary and nonstationary learning characteristics of the LMS adaptive filter," *Proc. IEEE*, vol. 64, pp. 1151–1162, 1976.

[2] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Prentice-Hall, Upper Saddle River, NJ, 1985.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins Univ. Press, Baltimore, MD, second edition, 1989.

[4] M. Effros and P. A. Chou, "Weighted universal transform coding: Universal image compression with the Karhunen-Loève transform," in *Proc. IEEE Int. Conf. Image Proc.*, 1995, vol. II, pp. 61–64.

[5] M. Effros, "Fast weighted universal transform coding: Toward optimal, low complexity bases for image compression," in *Proc. IEEE Data Compression Conf.*, J. A. Storer and M. Cohn, Eds., Snowbird, Utah, Mar. 1997, pp. 211–220, IEEE Comp. Soc. Press.

[6] J. Adler, B. D. Rao, and K. Kreutz-Delgado, "Comparison of basis selection methods," in *Proc. Asilomar Conf. on Sig., Sys. & Computers*, 1996.

[7] V. K Goyal, M. Vetterli, and N. T. Thao, "Quantized overcomplete expansions in $\mathbb{R}^N$: Analysis, synthesis, and algorithms," *IEEE Trans. Info. Theory*, vol. 44, no. 1, Jan. 1998.

[8] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge Univ. Press, 1985.

[9] T. W. Anderson, I. Olkin, and L. G. Underhill, "Generation of random orthogonal matrices," *SIAM J. Sci. Stat. Comput.*, vol. 8, no. 4, pp. 625–629, July 1987.

[10] M. Vidyasagar, *Nonlinear Systems Analysis*, Prentice Hall, Englewood Cliffs, NJ, 1978.

[11] P. M. Clarkson, *Optimal and Adaptive Signal Processing*, CRC Press, Boca Raton, FL, 1993.

[12] C. R. Johnson, Jr., *Lectures on Adaptive Parameter Estimation*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[13] V. K Goyal, J. Zhuang, and M. Vetterli, "Universal transform coding based on backward adaptation," in *Proc. IEEE Data Compression Conf.*, J. A. Storer and M. Cohn, Eds., Snowbird, Utah, Mar. 1997, pp. 231–240, IEEE Comp. Soc. Press.

[14] H. P. Kramer and M. V. Mathews, "A linear coding for transmitting a set of correlated signals," *IRE Trans. Info. Theory*, vol. 23, pp. 41–46, Sept. 1956.

[15] J. J. Y. Huang and P. M. Schultheiss, "Block quantization of correlated Gaussian random variables," *IEEE Trans. Comm.*, vol. 11, pp. 289–296, Sept. 1963.

[16] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Acad. Pub., Boston, MA, 1992.