Principal components: a descent algorithm

Rebeca Salas–Boni and Esteban G. Tabak *

February 11, 2014

Abstract

A descent procedure is proposed for the search of low-dimensional subspaces of a high-dimensional space that satisfy an optimality criterion. Specifically, the procedure is applied to finding the subspace spanned by the first m singular components of an n-dimensional dataset. The procedure minimizes the associated cost function through a series of orthogonal transformations, each represented economically as the exponential of a skew-symmetric matrix drawn from a low-dimensional space.

Keywords: Principal component analysis.

1 Introduction

Many frequently arising problems involve finding the small-dimensional subspace X of a larger space Z that minimizes a functional $f(X)$. Probably the most ubiquitous among these is the search of principal components: the m-dimensional subspace of R^n that best captures the variability of a set z of N observations. A closely related search is that of the subspace spanned by the eigenvectors corresponding to the m largest eigenvalues of a positive definite matrix C: when C is the empirical covariance matrix of the observations in the set z, the two problems are the same.

These are orthogonally constrained minimization problems: we minimize a cost function $f(Q_x)$, where Q_x is an $n \times m$ matrix such that ${Q_x}'Q_x = I_m$. For instance, the second problem above can be written in the form $\min_{Q_x} f(Q_x) = -\mathrm{tr}({Q_x}'CQ_x)$ [8]. Yet all these problems have a critical degeneracy: two orthogonal matrices Q_x are equivalent when they span the same column space X. In this work, we propose a methodology that exploits this degeneracy to yield a simple, yet very effective algorithm for the search for optimal subspaces. We focus on principal component analysis, because of its wide applicability and because its particular structure allows for extra simplifications.

A very general and powerful geometric view of optimization with orthogonality constraints has been proposed in [3]. Here the degeneracy mentioned above is characterized in terms of the Grasmann manifold, the quotient space of the Stiegel manifold of orthonormal $n \times m$ matrices

*Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012, USA, salasboni@cims.nyu.edu, tabak@cims.nyu.edu.

with respect to the group of orthogonal transformations in their column space. The algorithm of this article can be framed within this general viewpoint, and we have tried to refer to the appropriate geometrical concepts throughout our discussion. Yet the methodology that we propose does not really require the language and tools of differential geometry, which though elegant and powerful, may obscure the conceptual simplicity of the proposed algorithm to the non-specialist.

The main ingredients of the new algorithm are the completion of the matrix Q_x into an $n \times n$ orthogonal matrix $Q = [Q_x Q_y]$, the factorization of Q into simpler orthogonal matrices Q_n, and the expression of each of these as the exponential of a skew-symmetric matrix A_n. The degeneracy in the choice of Q results in a particularly sparse block structure for A_n, which renders its exponentiation and the composition of the Q_n's much less computationally expensive that they would be otherwise. In addition, the particularly simple form of the objective function for principal components results in a very inexpensive implementation of second-order descent on the entries of A_n.

A similar use of a matrix exponential representation for orthogonal matrices has been applied in [9] to minimize the maximum eigenvalue of a family of symmetric matrices $A(x)$. Their methodology allows for second-order descent –i.e., Newton's method– even when the sought eigenvalue has multiplicity greater than one. In the general geometric view of [3, 2], minimization of the cost function in the manifold of orthogonal matrices is achieved by moving towards the optimum along geodesics, which are described by the exponential map. In order to reduce computational costs, a truncated Taylor series expansion of the matrix exponential has been used in [1] to perform an approximate steepest descent.

A straightforward but computationally intensive way to compute singular components is through the singular value decomposition (SVD) of the n-by-N matrix z of observations:

$$z = U \Sigma V^*$$

where U is an n-by-n orthogonal matrix, Σ is a n-by-N diagonal matrix with real entries $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_N \geq 0$, and V a N-by-N orthogonal matrix. The calculation of the SVD requires $\mathcal{O}(Nn^2)$ flops [11]. The first m columns of U are the principal components sought. The singular value decomposition also provides us with the optimal low-rank (m) approximation to a general $n \times N$ matrix z:

$$z \approx U_m \Sigma_m V_m^*$$

where the columns of U_m and V_m are the first m columns of U and V respectively, and Σ_m is a diagonal matrix containing the m largest singular values of z. Then

$$\| z - U_m \Sigma_m V_m^* \| = \sigma_{m+1}$$

where σ_{m+1} is the $(m+1)$-th singular value of z, and $\| \cdot \|$ is the spectral norm.

Among the novel alternative approaches for finding low-rank approximations to matrices, randomized algorithms [4, 10] compute, with high probability, very accurate approximations to matrices of arbitrary size.

2 The problem

Let z denote the data set of observations $z_j \in R^n,\ j = 1, \ldots, N$. From this set, one seeks the hyperplane through the origin of given dimension $m < n$ that best captures the variability of z. In typical applications to dimensional reduction, m is much smaller than n, seeking either data compression for storage or transmission, or a small-dimensional manifold where the phenomena underlying the observational set z can be more easily understood. In terms of the operator P that projects onto the hyperplane, the cost function to minimize is

$$\min_P c = \min_P \frac{1}{2N} \sum_{j=1}^{N} \|z_j - P(z_j)\|^2, \tag{2.1}$$

the 2-norm of the distance between the points and their projection. The hyperplane sought can be characterized by a set of m orthonormal vectors spanning it. Writing these vectors as columns of a matrix Q_x, the projection operator P acquires the matrix representation

$$P = Q_x {Q_x}'.$$

A vector $x_j \in R^m$ can be assigned to the projection $P(z_j) \in R^n$ through

$$x_j = {Q_x}' z_j, \quad P(z_j) = Q_x x_j.$$

Furthermore, we can introduce coordinates $y_j \in R^{n-m}$ in the orthogonal complement to x_j, through

$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} = \begin{bmatrix} Q_x' \\ Q_y' \end{bmatrix} z_j = Q' z_j, \quad x_j \in R^m, \quad y_j \in R^{n-m}, \quad Q = [Q_x Q_y] \text{ orthogonal.}$$

In terms of these, the minimization problem (2.1) adopts the form

$$\min_Q c = \min_Q \frac{1}{2} \overline{\|y_j\|^2}, \tag{2.2}$$

where the bar stands for averaging over the N observations.

Alternatively, one can introduce the empirical covariance matrix

$$C \in R^{n \times n}, \quad C = \frac{1}{N} \sum_j z_j \cdot z_j'$$

where $z_j \cdot z_j'$ is the $n \times n$ matrix resulting from the outer product of the vector z_j. C transforms, under orthogonal maps Q as above, through

$$C \to Q' C Q.$$

Partitioning C into four blocks, two for the covariance matrices of the x and y variables alone, and the other two for their cross-covariance,

$$C = \begin{bmatrix} C_x^x & C_x^y \\ C_x^{y\prime} & C_y^y \end{bmatrix}, \tag{2.3}$$

the problem becomes

$$\min_Q c = \min_Q \frac{1}{2}\mathrm{tr}\left(C_y^y\right). \tag{2.4}$$

Thus there are two equivalent formulations of the problem: in (2.2), one seeks the hyperplane spanned by the first m principal components of the matrix z; in (2.4), the hyperplane of the first m eigenvectors of the non-negative definite matrix C. The second formulation is more compact, since the individual observations are gone from the picture. The first, however, lends itself to useful generalizations, such as the search for time-dependent or nonlinear principal components. Either formulation can be handled by the algorithm described below.

2.1 Retrieving the principal components of z

Our algorithm outputs in the columns of Q_x the subspace spanned by the first m principal components of the n-by-N matrix of observations z. However, one might be interested in obtaining the principal components themselves. Let $z = USV'$ denote the SVD decomposition of z, and C the n-by-n covariance matrix. After we have applied our algorithm and obtained the matrix Q_x, we set $\tilde{C}_x^x = Q_x' C Q_x$. Since $\tilde{C}_x^x$ is an m-by-m positive definite matrix with $m << n$, we can find unexpensively by conventional methods its SVD decomposition,

$$\tilde{C}_x^x = U_{\tilde{C}} S_{\tilde{C}} V_{\tilde{C}}',$$

From this decomposition, we can retrieve the first m singular components and values of the SVD decomposition of z:

$$U_m = Q_x U_{\tilde{C}},$$

the first m singular values of z through

$$S_m = \sqrt{N}\left(\sqrt{S_{\tilde{C}}}\right)$$

and the first m columns of V through

$$V_m = S_m^{-1} U_m' z$$

3 The algorithm

We propose an iterative algorithm that writes the desired orthogonal transformation Q as the composition of simpler orthogonal maps Q_k:

$$Q = \ldots Q_3 Q_2 Q_1 Q_0.$$

Thus, in each step, one has

$$z^k = Q_k z^{k-1}, \quad z^0 = Q_0 z,$$

where Q_0 is a pre-conditioning orthogonal matrix to be described below, and the super-script k denotes the number of the iteration. Equivalently, in terms of the covariance matrix C,

$$C_k = Q_k C_{k-1} {Q_k}'.$$

After a sufficient number of iterations, the first m rows in z^k converge to the N observations projected onto the subspace spanned by the first m singular vectors of z, and C_k approximates the covariance matrix whose m-by-m submatrix C_x^x represents the covariance of those m-dimensional projections.

Orthogonal matrices $Q \in R^{n\times n}$ depend on $\frac{1}{2}n \times (n-1)$ parameters. The simplest way to make this dependence explicit is to write Q_k as the exponential of a skew-symmetric matrix A_k:[1]

$$Q_k = e^{A_k}, \quad {A_k}' = -A_k.$$

In the limit of infinitesimal rotations, the (i,j) entry of A_k represents the angle of rotation in the plane spanned by the z_i and z_j coordinates. Yet any rotation in a plane spanned by two x or two y coordinates alone would only re-parameterize the subspaces spanned by the columns of Q_x or Q_y without changing them. Hence we may restrict consideration to matrices A_k of the block form

$$A_k = \left(\begin{array}{c|c} 0 & S_k \\ \hline -{S_k}' & 0 \end{array}\right), \quad S_k \in R^{m\times(n-m)},$$

describing only rotations with one leg in the x and one in the y subspaces. In the geometric language of [3], this corresponds to only making moves in the Stiegel manifold of orthonormal $n \times m$ matrices that are, to leading order, also moves in the Grasmann manifold.

Matrices of this form are easy to exponentiate, using the compact singular value decomposition of S_k:

$$S_k = \sum_{j=1}^{m^*} \sigma_j u_j v_j', \quad u_j \in R^m, \quad v_j \in R^{n-m},$$

where $m^* = \min(m, n-m)$. Then

$$Q_k = e^{A_k} = I + A_k + \frac{{A_k}^2}{2!} + \cdots = \left(\begin{array}{c|c} Q_{x,x} & Q_{x,y} \\ \hline Q_{y,x} & Q_{y,y} \end{array}\right) \tag{3.1}$$

where

$$Q_{x,x} = I + \frac{1}{2!}(-SS^\top) + \frac{1}{4!}(SS^\top SS^\top) + \frac{1}{6!}(-SS^\top SS^\top SS^\top) + \cdots = \sum_{j=1}^{m} u_j \cos(\sigma_j)\, u_j',$$

[1] This applies only those orthogonal matrices Q_k that are orientation preserving (i.e., have $\det(Q_k) = 1$), but this restriction is immaterial to our application, since it is only the sub-space spanned by the first m columns of Q_k that matters, not the individual vectors describing it or their sign.

Similarly, we derive

$$Q_{x,y} = \sum_{j=1}^{m} u_j \sin(\sigma_j)\, v_j', \qquad Q_{y,x} = -Q_{x,y}^\top$$

and

$$Q_{y,y} = \sum_{j=1}^{n} v_j \cos(\sigma_j)\, v_j' = I + \sum_{j=1}^{m} v_j(\cos(\sigma_j) - 1)v_j^\top .$$

Combining these identities, we obtain:

$$Q_k = \left(\begin{array}{c|c} Q_{x,x} & Q_{x,y} \\ \hline Q_{y,x} & Q_{y,y} \end{array}\right) = \begin{pmatrix} 0_{m^*} & 0 \\ 0 & I_{n-m^*} \end{pmatrix} + \sum_{j=1}^{m^*} \begin{pmatrix} u_j & 0 \\ 0 & v_j \end{pmatrix} \begin{pmatrix} \cos(\sigma_j) & \sin(\sigma_j) \\ -\sin(\sigma_j) & (\cos(\sigma_j)-1) \end{pmatrix} \begin{pmatrix} u_j' & 0 \\ 0 & v_j' \end{pmatrix}. \tag{3.2}$$

The interpretation of (3.2) is straightforward again in the limit of infinitesimal rotations, where the $m(n-m)$ rotations given by the entries of S_k –one for each pair of coordinates (x_i, y_j)– can be far more economically described by just m^* rotations, with angles given by the principal values σ_j of S_k, in the planes spanned by the corresponding left and right principal components.[2]

3.1 Gradient descent

In order to complete the description of the algorithm, we need only specify the matrix S_k to use in each step. This must be chosen so as to descend toward a minimal value of the cost c in (2.2) or (2.4). Notice that, up to here, no reference has been made to this cost, so the description so far applies to any problem where an m-dimensional subspace of an n-dimensional space is sought. From here on, we focus on the specific cost at hand, whose structure leads to further simplifications.

The simplest choice for S_k is through gradient descent:

$$S_k = -\alpha G, \tag{3.3}$$

where

$$G_i^j = \left.\frac{\partial c}{\partial S_i^j}\right|_{S_k=0}$$

is the gradient of the cost function c, $\alpha > 0$ is a –possibly adaptive– learning rate, and S_i^j denotes the i,j-th entry of the matrix S_k. Since S_i^j corresponds to the angle of rotation in the (x_i, y_j)-plane, we have that

$$\left.\frac{\partial y_k}{\partial S_i^j}\right|_{S_k=0} = -\delta_k^j x_i,$$

[2]The notation in (3.2), well-suited for computation, may not be so clear geometrically. To clarify it, decompose the identity matrix I_{n-m^*} into $\sum_{j=1}^{m^*} v_j {v_j}' + P$, where $P = I_{n-m^*} - \sum_{j=1}^{m^*} v_j {v_j}'$ represents the projection onto the subspace orthogonal to the first m^* principal components v_j. This has the effect of canceling out the ones subtracted from $\cos(\sigma_j)$ in (3.2), and replacing the I_{n-m^*} by the projection operator P, thus providing the geometrical picture described in the text.

so

$$G_i^j = \frac{1}{2}\frac{\partial \overline{\|y\|^2}}{\partial S_i^j}\bigg|_{S_k=0} = -\overline{x_i y_j},$$

or, in terms of (2.3),

$$G = -C_x^y.$$

We adopt two different learning rates α for the examples of gradient descent in section 6. The first one is given by

$$\alpha = \frac{\epsilon}{\sqrt{|G|^2 + \epsilon^2}}, \tag{3.4}$$

where

$$|G|^2 = \sum_{i,j} {G_i^j}^2$$

and ϵ is the desired size for rotations away from the minimal c:

$$|G| \gg \epsilon \Rightarrow |S_k| \approx \epsilon$$

Close to c_{min} the gradient G is small, and the steps must become correspondingly smaller, to avoid overshooting. The choice in (3.4) then yields steps of size

$$|S_k| \approx |G| \tag{3.5}$$

An alternative learning rate α, denoted by α^* in the numerical examples, uses Newton's method in the direction of the gradient:

$$\alpha = -\frac{\frac{\partial c}{\partial \alpha}}{\frac{\partial^2 c}{\partial \alpha^2}},$$

a method that we will refer to as quadratic gradient descent.

For any fixed vector v, we have

$$\frac{\partial c\,(\alpha v)}{\partial \alpha} = v \cdot G,$$

where G is the gradient of c and the dot product between two matrices A and B is defined as $\langle A, B\rangle = \mathrm{tr}\,(A'B)$. Similarly,

$$\frac{\partial^2 c}{\partial \alpha^2} = v \cdot Hv,$$

where H is the Hessian of c. Hence, along the direction $v = G$ of the gradient,

$$\begin{aligned} \alpha^* &= -\frac{\frac{\partial c}{\partial \alpha}}{\frac{\partial^2 c}{\partial \alpha^2}} = -\frac{G \cdot G}{G \cdot HG} \\ &= -\frac{C_x^y \cdot C_x^y}{C_x^y \cdot (C_x^x C_x^y - C_x^y C_y^y)}, \end{aligned} \tag{3.6}$$

using an expression for H derived below.

3.2 Second order descent

In order to improve the rate of convergence of the algorithm beyond that of gradient descent, one can introduce the Hessian

$$H_{i_1 j_1}^{i_2 j_2} = \left. \frac{\partial^2 c}{\partial S_{i_1}^{j_1} S_{i_2}^{j_2}} \right|_{S_k = 0}$$

and use Newton's method, solving for S_k the system

$$\sum_{hl} H_{ij}^{hl} S_h^l = -G_i^j. \tag{3.7}$$

This appears at first sight to be quite costly, since the system in (3.7) has $m(n-m)$ equations and unknowns. Yet the particular structure of the cost function comes to our help: the Hessian for c from (2.2) has entries

$$H_{i_1 j_1}^{i_2 j_2} = \delta_{j_1}^{j_2} \overline{x_{i_1} x_{i_2}} - \delta_{i_1}^{i_2} \overline{y_{j_1} y_{j_2}},$$

which allows us to write the system (3.7) in the simple matrix form

$$C_x^x S_n - S_n C_y^y = C_x^y. \tag{3.8}$$

Both $C_x^x \in R^{m \times m}$ and $C_y^y \in R^{(n-m)\times(n-m)}$ are positive definite matrices and, near convergence, all eigenvalues of C_x^x are larger than those of C_y^y. This implies the existence of a unique solution to (3.8), given by

$$S_n = U Z V',$$

where

$$C_x^x = U \Lambda U', \quad C_y^y = V \Gamma V',$$

with U and V orthogonal and Λ and Γ diagonal matrices, and

$$Z_i^j = \frac{(U' C_x^y V')_i^j}{\Lambda_i^i - \Gamma_j^j}.$$

Using this explicit formula involves diagonalizing C_y^y, a prohibitively expensive task. However, the fact that the Λ_i^i's are larger than the Γ_j^j's implies also the convergence of the following iterative algorithm to find S_k:

$$\begin{aligned} S_k^0 &= 0 \\ S_k^{h+1} &= (C_x^x)^{-1} \left(C_x^y + S_k^h C_y^y \right), \quad h = 0, 1, 2, \cdots \end{aligned} \tag{3.9}$$

which involves solving systems with only m unknowns.

This ordering between Λ_i^i's and Γ_j^j's holds near the optimum, since the cost function c is precisely the trace of C_y^y: if any eigenvalue of C_x^x were smaller than one of C_y^y, exchanging the corresponding eigenvectors between the x and y subspaces would produce a smaller cost.

Yet this criterion for the convergence of (3.9) needs not hold at the onset of the algorithm. To address this, one can resort to a simple pre-conditioning step, sorting the variables by their individual variance. Thus Q_0 is a permutation matrix that sorts the entries on the main diagonal of the covariance matrix C in decreasing order. To further precondition, one could take a few steps of gradient descent. More effective, however, is to use, for a small number of steps k_p, (3.9) with only one iteration:

$$S_k = (C_x^x)^{-1} C_x^y, \quad k \leq k_p. \tag{3.10}$$

Since C_x^x is positive definite, the resulting S_k lies in a direction in which the cost c decreases: $(S_k, G) < 0$. In fact, (3.10) can be used throughout in lieu of (3.9), since C_x^x is a robust surrogate for the full Hessian H. The rate of convergence is, of course, worse than by using (3.9). On the other hand, the computational cost per step is much cheaper, particularly if one is using the matrix of observations Z rather than the covariance C: there is no longer need to compute C_y^y in each step, the most costly component of the covariance.

A more robust approach that also handles the situation where there is no gap between the smallest eigenvalue of C_x^x and the largest eigenvalue of C_y^y, is to mollify the Hessian through the addition of a constant matrix:

$$H \to H + \epsilon,$$

where $\epsilon > 0$ is a suitable mollification parameter interpolating between Newton's method ($\epsilon \to 0$) and gradient descent ($\epsilon \to \infty$, with learning rate $1/\epsilon$). Then the iterative scheme in (3.9) becomes

$$\begin{aligned} S_k^0 &= 0 \\ S_k^{h+1} &= (C_x^x + \epsilon I)^{-1} \left(C_x^y + S_k^h \left(C_y^y - \epsilon I \right) \right), \quad h = 0, 1, 2, \cdots \end{aligned} \tag{3.11}$$

The mollified Hessian is positive definite whenever the largest eigenvalue of C_y^y is smaller that the smallest eigenvalue of C_x^x plus 2ϵ. Thus ϵ can be chosen adaptably, or made to decrease as the algorithm progresses. In the numerical examples presented below, we have implemented the simplest choice of a fixed ϵ.

4 Computational complexity

There are two possible implementations of the algorithm: one in terms of the $n \times N$ matrix z of observations, and the other in terms of the $n \times n$ covariance matrix C. If one is provided with z and chooses to implement the algorithm in terms of C, computing $C = zz'$ requires $O(n^2 N)$ flops. Other than this and the cost of the preconditioning steps, negligible in comparison to the others, we need to find the cost of each step in the iteration. We describe these steps in Algorithm 1.

In terms of C, finding S_k though gradient descent, (3.3), requires $O(m^2)$ flops, through the approximate second order in (3.10), $O(m^3) + O(m(n-m))$ flops, and, through the full second order procedure in either (3.9) or (3.11), $O\left(m(n-m)^2\right)$ flops.

Algorithm 1 For k=1 until convergence

1. Find S_k though (3.3), (3.10) or (3.9).
2. Exponentiate A_k to form Q_k.
3. Update C through $C_k = Q_k C_{k-1} Q_k{}'$, or z though $z_k = Q_k z_{k-1}$.

The exponentiation of A_k requires finding the compact singular value decomposition of S_k, or $O\left(m(n-m)^2\right)$ flops. Because of the structure of the resulting Q_k from (3.2), multiplying it on either right or left by an n-dimensional vector involves $O(mn)$ operations, so updating z requires $O(mnN)$ flops, while updating C requires $O\left(mn^2\right)$ flops.

In terms of z, we need to add to these the $O(m(n-m)N)$ flops of the computation of C_x^y for (3.3) and (3.10), or the $O(n^2N)$ flops of computing all of C for (3.9/3.11).

As for the required number of steps, it depends on the accuracy sought and the methodology chosen for descent: if (3.9) is used, the rate of convergence is super-quadratic, so the number of steps grows at worst logarithmically with the problem's size. This still hold approximately for (3.10) if only a reasonable level of accuracy is sought. If, on the other hand, one seeks to capture variability up to a small fraction of the difference between the m-th and $m+1$-th singular values of z, then the latter alternative requires substantially more steps than the former. Regarding the number of iterations required in (3.9/3.11), see the examples below.

In summary, when the algorithm is implemented in terms of the covariance matrix C, the full Hessian should be used for descent, and the resulting total work per step is $O\left(mn^2\right)$, with a number of steps that depends only weakly on the problem's size. This still holds for the procedure in terms of z when the number of observations N is comparable or smaller than n, the total number of variables. When $N > n$, it may be worth trading a larger total number of steps for the lighter work per step, $O(mnN)$, of using (3.10) for descent.

5 Alternative procedures

The most costly component of the procedure described is the computation of the compact singular value decomposition of S_k: even though this involves only $m \ll n$ principal components, m can still be large. There are at least two possible approaches to reducing this cost. The most straightforward one is to carry out the singular value decomposition only approximately. For instance, one may seek just the first $m' \ll m$ principal components of S_k, using recursively the algorithm of this article.

A more radical approach is to propose, at each step, a matrix S_k of the form

$$S_k = \sum_{j=1}^{m} \sigma_j u_j v_j',$$

where the u_j's and v_j's are prescribed orthonormal vectors of dimension m and $(n-m)$

respectively: instead of computing the optimal S_k among general $m \times (n-m)$ matrices and then finding its singular value decomposition, we are limiting our search to matrices with prescribed principal components, but free singular values σ_j. It is easy to see that the optimal σ's satisfy the following projected version of (3.8):

$$\sigma_j \left[u_j' C_x^x u_j - v_j' C_y^y v_j \right] = u_j' C_x^y v_j, \tag{5.1}$$

a single scalar equation for each σ_j that requires no iteration. Moreover, the resulting S_k has a known singular value decomposition, thus reducing the computational cost of each step to the calculation of $O(m)$ products of matrices times vectors.

There are various alternatives on how to select the singular components u_j and v_j, each involving a trade off between the computational cost of each step and the global rate of convergence. A thorough exploration of the possible strategies, however, lies beyond the scope of this paper.

6 Numerical examples

In this section, we illustrate the performance of the algorithm presented in this paper via several numerical examples, some synthetic and one using oceanographic data.

First, we apply our methodology to two classes of matrices with singular values decaying in ways that have been deemed typical in practice [10]: exponential in one class, and exponential followed by linear decay in the other. We also experiment with the gap between the m-th and $(m+1)$-th singular values. To build the covariance matrix C_0, we propose a diagonal matrix Σ_0 of singular values with the desired properties, randomly draw an orthogonal matrix Q_0, and define $C_0 = Q_0 \Sigma_0 Q_0'$.

We monitor the convergence to the hyperplane sought through two quantities: the cost function $c = tr(C_y^y)$, which should converge to its true value given by

$$c_{\text{true}} = \sum_{k=m+1}^{n} (\Sigma_0)_k^k \,,$$

and

$$e_Q = \frac{1}{\sqrt{m}} \| Q_x' (I - (Q_0)_x \, (Q_0)_x') \|_F, \tag{6.1}$$

the Frobenius norm of the difference between the subspace spanned by the m columns of the matrix Q_x computed by the algorithm and its projection over the subspace X, spanned by the columns of the true $(Q_0)_x$.

The first class has exponentially decaying singular values given by

$$\sigma_j = \sigma_m^{j/m}, \quad j = 1 \ldots n, \tag{6.2}$$

while, for the second class, the singular values are chosen in such a way that the first m values decay exponentially and the remaining ones linearly [10]:

$$\sigma_j = \begin{cases} (\sigma_m)^{(j/m)} \,, & \text{for } j \le m \\ \sigma_m \cdot \left(\frac{m}{j} \right) , & \text{for } j > m. \end{cases} \,. \tag{6.3}$$

In both cases, we have adopted $\sigma_m = .01$. In order to study the effect of the relative gap between the mth and $(m+1)$th singular values, we have also studied the following example:

$$\sigma_j = \begin{cases} (\sigma_m)^{(j/m)}, & \text{for } j \leq m \\ \alpha\sigma_m \cdot \left(\frac{m+1}{j}\right), & \text{for } j > m. \end{cases} \tag{6.4}$$

where $\alpha \leq 1$.

Our first set of experiments takes data from (6.2), with $m = 32$ and $n = 512$, and compares the performance of four candidate procedures. The first two correspond to gradient and quadratic gradient descent, with learning rates α from (3.3) and (3.6) respectively. The last two methods are second order descent, one using a surrogate and the other the full Hessian. The four approaches are sorted by their computational cost per step.

1. Straightforward gradient descent, with the maximal learning rate in (3.3) set to $\epsilon = 0.1$,
2. Quadratic gradient descent (3.6),
3. Using C_x^x as a surrogate for the full Hessian in (3.10),
4. Applying the full second order procedure in (3.9), with 100 iterations per step, after 10 preconditioning steps that use just the C_x^x component of the Hessian as before.

The number of preconditioning steps was determined heuristically after trying values ranging from 1 to 50 across different experiments. As we can see in the results displayed in figure 1, all four procedures converge, but at very different rates, with the full second-order descent converging to machine precision in just three steps after the preconditioning.

The second set of experiments takes data from (6.4) with $m = 32$ and $n = 512$, and three values of α: the degenerate case with $\alpha = 1$, with no gap between the last resolved and first unresolved singular values, $\alpha = 0.9$, with a small gap, and $\alpha = 0.5$, with a significant gap. This experiment was designed to explore the behavior of our algorithm when facing slowly decaying eigenvalues. The results, displayed in figure 2, show the effect of increasing the number of steps h in the iterative procedure (3.9) for inverting the Hessian. With a large spectral gap, just one iteration per step does a perfect job. For the more challenging case of $\alpha = 0.9$, the number of iterations h affect the global rate of convergence; with about $h = 50$ iterations we have again full convergence in just three steps. The situation is similar for the degenerate case with $\alpha = 1$, where the algorithm is even more sensitive to the number of iterations h used to approximate the Hessian. Notice though that the rate of convergence is comparatively slow only once the error is far below the $m+1$ singular value (0.01), i.e. when any further decrease of the cost function represents only a small fraction of the true cost.

Figure 3 shows how this behavior changes under the mollification in (3.11) with two values of ϵ: 0.01 and 0.001. As one can see, the smaller value has little effect, while the effect of the larger mollification is, when the gap between wanted and unwanted eigenvalues is small or null, to make the number of iterations h irrelevant, so $h = 1$ is enough, at the expense of a slower convergence rate.

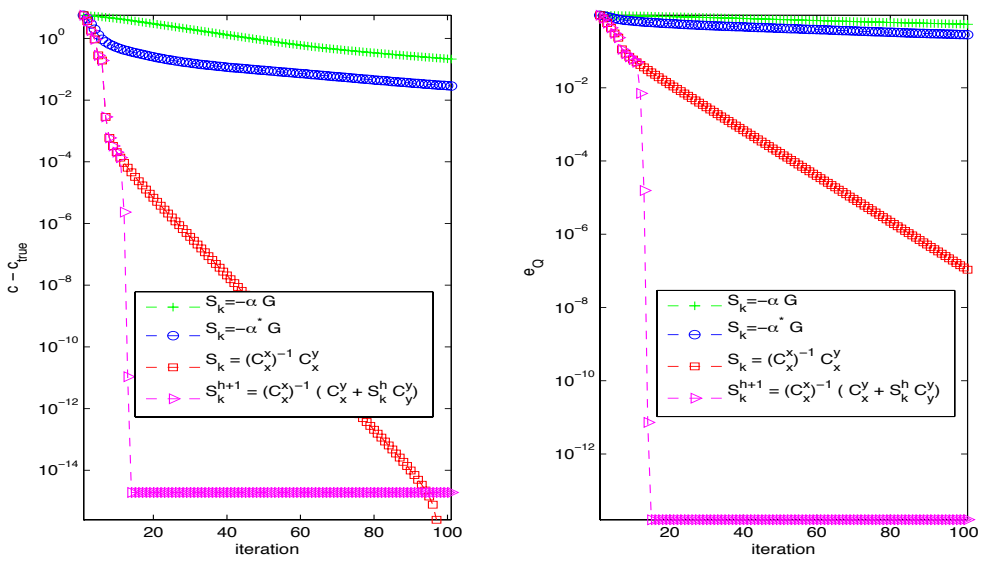

Figure 1: Comparison of gradient descent with both learning rates α, the use of C_x^x as a surrogate Hessian, and of the full Hessian. The data has exponentially decaying singular values, as in (6.2), with $\sigma_m = .01$. A subspace of dimension $m = 32$ is sought in a space of dimension $n = 512$. On the left, a logarithmic plot of the cost function as a function of the step; on the right, the measure e_Q from (6.1) of the error in the determination of the subspace X. Notice the very fast convergence of the fully second-order procedure, in magenta, which takes just three steps after the preconditioning to reach machine precision. The pseudo-second-order procedure in (3.10), in red, is a strong competitor, balancing a slower rate of convergence with a much reduced work per step. As expected, both gradient descent methods are considerably slower. However, the quadratic gradient descent with learning rate α^* from (3.6), in blue, outperforms the regular gradient descent with α from (3.4), in green.

The third set of experiments, displayed in figure 4, uses data from (6.3), and shows the effect that the problem's size, through the dimensions m and n, has on the rate of convergence. Here again 10 preconditioning steps are followed by the inversion of the full Hessian, through (3.9) with 100 iterations. The results indicate that the dimension n of the original space Z of observations has nearly no effect on the rate of convergence, while the effect of the dimension m of the target subspace X sought is more pronounced, though also minimal.

6.1 Application to sea surface temperatures

To assess the performance of the descent procedure on a real-life application, we chose a database comprising extended reconstructed global sea surface temperatures based on

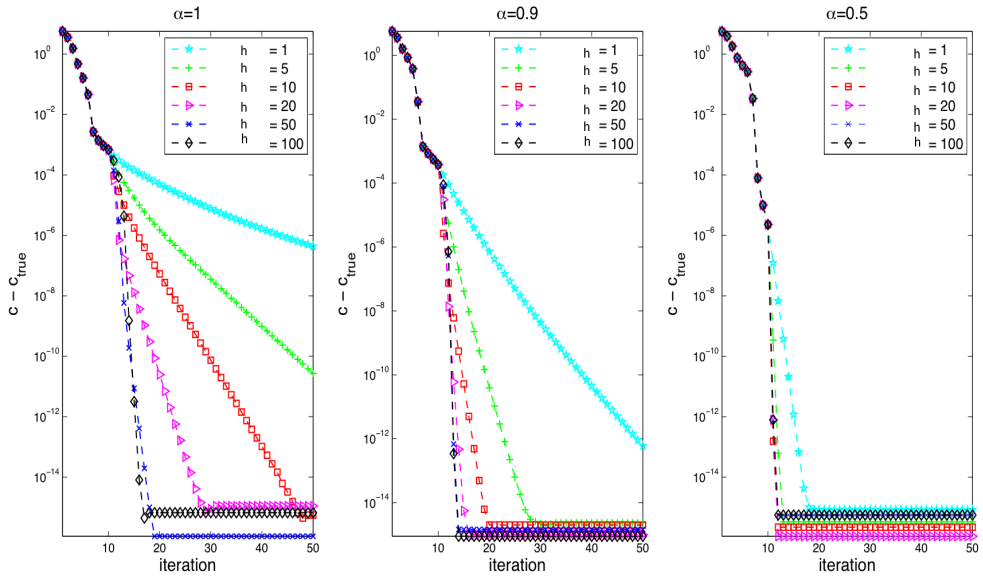

Figure 2: Dependence of the rate of convergence of the second order algorithm on h, the number of iterations in (3.9). The data have the form in (6.4) with $\sigma_m = .01$, $m = 32$ and $n = 512$. The plot on the left corresponds to the case with $\alpha = 1$, with no gap between the m-th and $(m+1)$-th singular values. In this degenerate case, the solution for the optimal manifold X is not unique. The algorithm still converges to an optimal manifold, but more slowly that in the other cases. On the right, the case with $\alpha = 0.5$, a large gap between the last resolved and first unresolved singular value. Here a few iterations suffice, since $(C_x^x)^{-1} C_y^y$ is small. In the middle, a more regular case, not degenerate but with no significant gap between the resolved and unresolved part of the spectrum. In all cases, a sufficient number h of iterations per step is enough to yield convergence in just a handful of steps after the initial 10 for preconditioning. However, the larger the gap between the consecutive singular values, the less sensitive the algorithm is to the number of iterations h used to approximate the Hessian.

COADS data [6]. From these monthly averages, ranging from January 1854 to June 2005, we isolated those corresponding to the month of December. The data is provided on a 180-by-89 grid, a resolution of 2 degrees in both latitude and longitude. After discarding those points that lie over land, we are left with 150 records of December temperatures with $11,074$ points each. We subtract the climatology of each point on the sea surface –that is, its average December temperature over the 150 years– and store the adjusted data in the n-by-N matrix z, where $n = 11,074$ and $N = 150$. We seek the first three principal components of z. To this end, instead of working with the high-dimensional matrix $C = zz' \in R^{n\times n}$, we apply our algorithm to the far more compact $C^* = z'z \in R^{N\times N}$. After retrieving the principal

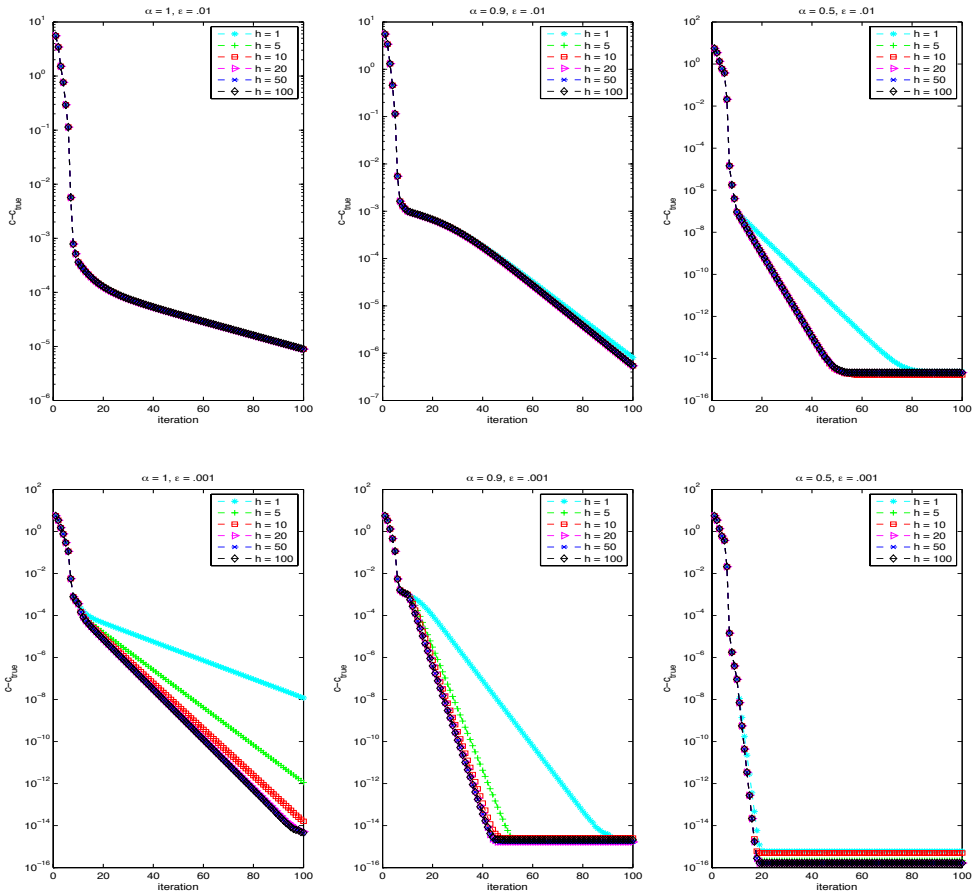

Figure 3: Dependence of the rate of convergence of the second order algorithm on h, the number of iterations in (3.9). In these two examples, we apply the mollification in (3.11) with two values of ϵ: 0.01 and ϵ: 0.001.

components of $C*$ using the procedure described in Section 2.1, we extract the principal components of the matrix we were originally interested in, C: each eigenvector u of C can be recovered from the corresponding eigenvector v of C^* with eigenvalue σ^2 through

$$u = \frac{1}{\sigma} z v.$$

Figure 5 shows the convergence of the algorithm; here the true singular values and principal components needed to evaluate the error at each step were computed independently through a standard MATLAB routine. For this real data, convergence to machine precision took place just one step beyond preconditioning. Figure 6 displays the first three principal components on a map: each location corresponds to one component of the singular vector, with its value

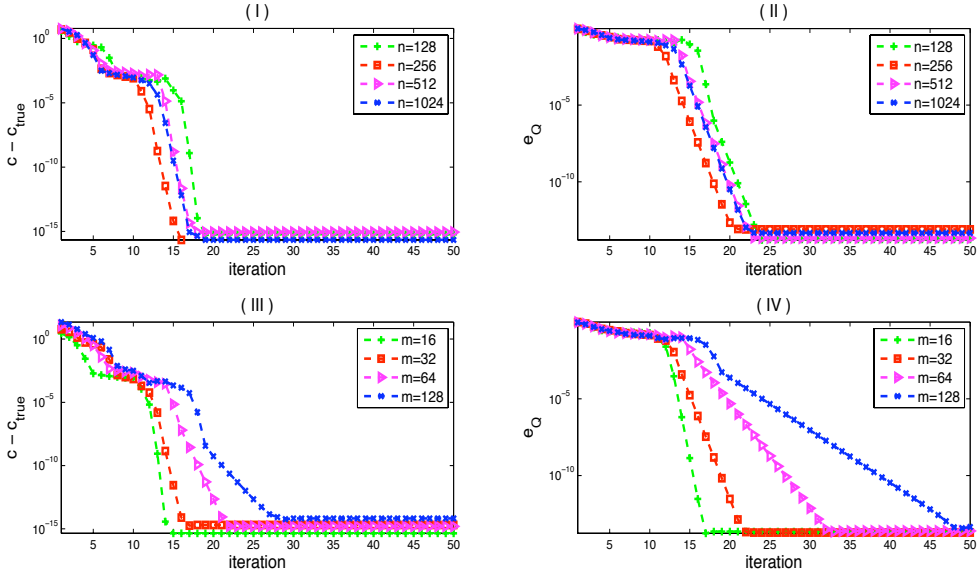

Figure 4: Effect of the problem's size on the rate of convergence. The singular values are computed as in (6.3) with $\sigma_m = .01$. In the top figures, the dimension of the subspace X sought is fixed at $m = 32$ while the dimension n of the full space Z is varied over an order of magnitude. In the bottom figures, the reverse is true: n is fixed at $n = 1024$ while m is varied. The rate of convergence shows almost no sensitivity to n, and little to m.

represented through color grading. In atmosphere and ocean science, these are referred to as Empirical Orthogonal Functions (EOFs) [7] and often associated with patterns of climate variability. The variability associated with El Niño Southern Oscillation, for instance, is easily identifiable as the second principal component.

7 Extensions

This article is mainly concerned with the computation of principal components, yet much of the methodology developed can be extended to a variety of scenarios. In this concluding section, we briefly sketch how some of these extensions might proceed. We discuss only those extensions where the manifold X sought is linear; extensions to curved manifolds, such as principal curves and surfaces, will be developed elsewhere.

The basic elements of the procedure, as described in section 3, apply to any situation where an m-dimensional subspace of an n-dimensional space is sought: the factorization of Q into the composition of simpler orthogonal matrices Q_k, the expression of the latter as exponentials of a matrix A_k with only $m \times (n - m)$ independent entries, and the reduction of the exponentiation process to the calculation of a small-dimensional singular value

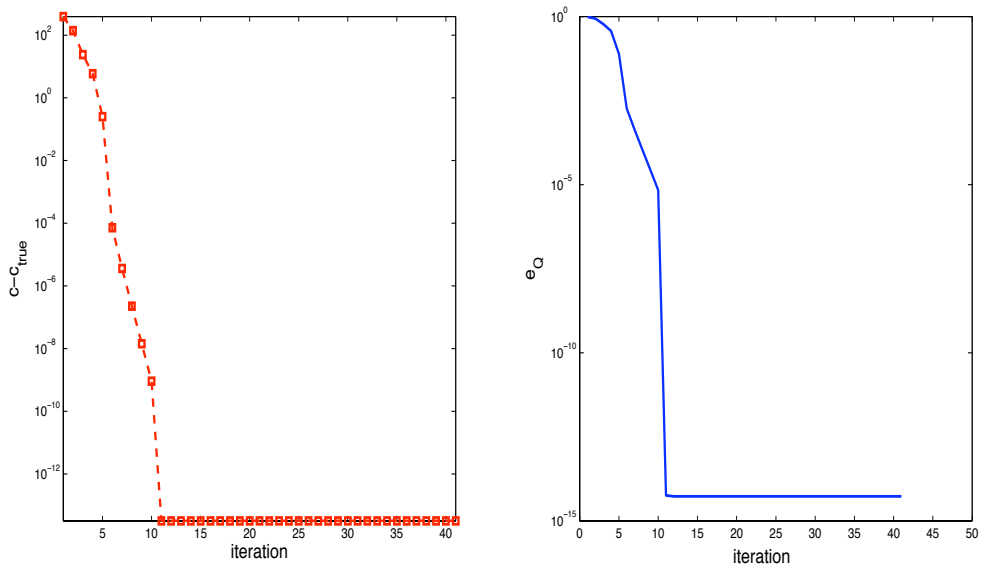

Figure 5: Search for the first three spacial principal components of the December sea-surface temperature field. On these real data, the convergence of the second order procedure to machine precision is nearly instantaneous, requiring just one step beyond the 10 used for preconditioning.

decomposition. Hence similar procedures apply to the following problems:

- **Non-autonomous principal components:** When the observations $z_j = z(t_j)$ represent a time series, the subspace of principal components X sought may depend on time: for instance, if in the oceanographic application above, we would have considered all months, not just December, we should have sought month-dependent principal components, with a period of one year. The procedure developed in this paper adapts easily to this scenario: one can seek a time-dependent orthogonal transformation $Q(t)$ by proposing, for each $Q_k(t)$, a transformation of the form

$$Q_k(t) = e^{A_k(t)}, \quad A_k(t) = f_k(t)\tilde{A}_k,$$

  where $\tilde{A}_k$ is a time-independent skew-symmetric matrix, and $f_k(t)$ is a prescribed function of t. The idea, pursued in [5] for principal dynamical components –see the item below–, is that any time-dependent orthogonal transformation can be factorized into Q_k's of this form. Each $f_k(t)$ works as a building block for more general time-dependent functions. The f_k's are typically simple functions, such as smooth localized bumps, with properties that reflect those required from $Q(t)$, such as periodicity or a minimal resolved scale. The only change in the cost function is the appearance of the $f_k(t_j)$'s as weights for the various observations z_j.

- **Principal dynamical components:** Again for time series $z_j = z(t_j)$, regular principal components do not yield any dynamical information, since all observations are treated as independent samples of an underlying random process. A more natural manifold X to seek, proposed in [5] and labeled "principal dynamical components", is the one that minimizes not the static information loss from considering only the projection $x = Q_x' z$, but rather the dynamical loss in the predictability of future events from x alone. In the simplest case of first-order autonomous Markov processes, the corresponding cost function is

$$c = \sum_{j=1}^{N-1} \left\| z_{j+1} - Q \begin{pmatrix} AQ_x{}' z_j \\ 0 \end{pmatrix} \right\|^2 = \sum_{j=1}^{N-1} \left\| \begin{pmatrix} x_{j+1} - Ax_j \\ y_{j+1} \end{pmatrix} \right\|^2 , \tag{7.1}$$

  depending now not only on the transformation Q but also on a predictive linear model given by the $m \times m$ matrix A. Minimizing c over A for Q given is a standard regression problem in m dimensions; minimizing c over Q for A given, on the other hand, can be handled by the algorithm of this article, with the cost function given by (7.1). Notice that this cost adds to the variance of the y variables, also present in regular principal components, the variance of the fraction of the x variables not accounted for by the predictive scheme represented by the matrix A.

8 Conclusions

A descent algorithm was developed for the calculation of principal components and, more generally, for the search of low-dimensional subspaces of a high-dimensional vector space satisfying an optimality property. The algorithm minimizes the cost function through a series of orthogonal rotations, each represented as the exponential of a skew-symmetric matrix picked from a comparatively small-dimensional manifold. The procedure is conceptually simple, easily extendable to different scenarios, and computationally effective, as demonstrated through a series of test cases.

9 Acknowledgements

The work of Rebeca Salas-Boni is partially supported by the CONACyT and NYU's GSAS MacCracken Fellowship, and the work of E. G. Tabak is partially supported by the Division of Mathematical Sciences of the National Science Foundation.

References

[1] T. Abrudan, J. Eriksson and V. Koivunen, *Optimization under unitary matrix constraint using approximate matrix exponential*. Signals, Systems and Computers, 2005. Conference Record Thirty-Ninth Asilomar Conference on Signals, Systems and Computers pp. 242–246

[2] P.A. Absil, R. Mahony and R. Sepulchre *Optimization algorithms on matrix manifolds*, Princeton University Press, 2008.

[3] A. Edelman, T. A. Arias and S. T. Smith , *The geometry of algorithms with orthogonality constraints*, SIAM J. Matrix Anal. Appl., 20 (1998), pp. 303–353.

[4] N. Halko, P. G. Martinsson and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.* arXiv, 2009.

[5] M. D. de la Iglesia and E. G. Tabak, *Principal dynamical components*, to appear in Comm. Pure Appl. Math, 2012.

[6] The International Research Institute for Climate and Society, http://iridl.ldeo.columbia.edu/SOURCES/.NOAA/.NCDC/.ERSST/

[7] E. N. Lorenz, *Empirical orthogonal functions and statistical weather prediction*, Sci. Rep. No. 1, Statist. Forecasting Proj., Dept. Meteor., MIT, 1956.

[8] J. H. Manton, *Optimization algorithms exploiting unitary constraints*, IEEE Trans.s on Signal Process., 50 (2002), pp. 635–650.

[9] M. L. Overton and R. S. Womersley, *Second derivatives for optimizing eigenvalues of symmetric matrices*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 697–718.

[10] V. Rokhlin, A. Szlam and M. Tygert, *A randomized algorithm for principal component analysis*, SIAM J. Matrix Anal. Appl., 31 (2009), pp .1100–1124.

[11] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM: Society for Industrial and Applied Mathematics, 1997.

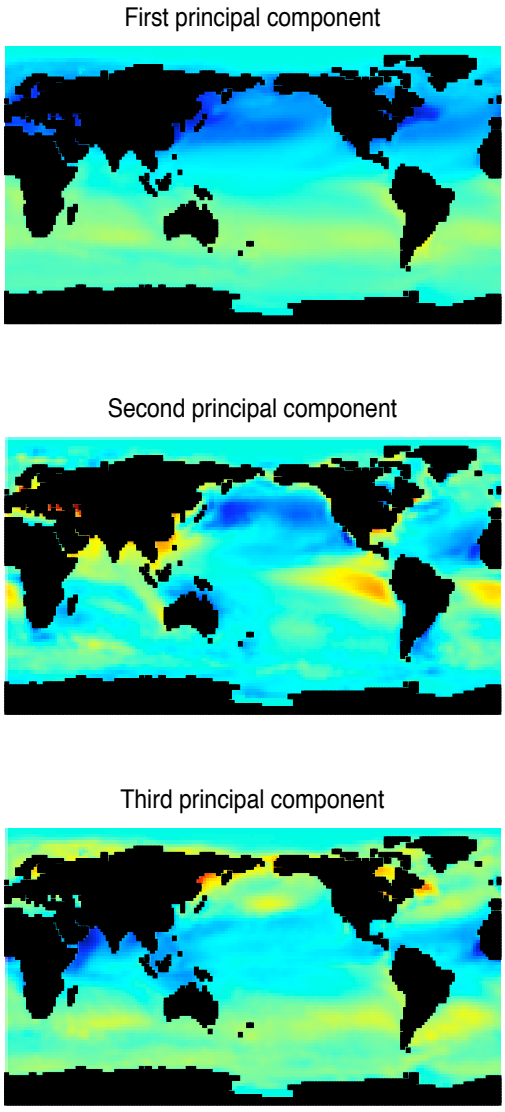

Figure 6: First three principal components of the December sea-surface temperature field, displaying values by color intensity on the map. These are the Empirical Orthogonal Functions of atmosphere and ocean science. The second component may be taken as representing the variability associated with El Niño Southern Oscillation, with its typical pronounced warming in the Eastern Pacific, offshore the coast of Perú. The corresponding singular values $\sigma_{1,2,3}$ associated with these three components are 19.75, 3.74 and 2.91 respectively.