# An Introduction to Principal Component Analysis and Online Singular Value Decomposition

Greg Coombe

October 31, 2006

## 1 Abstract

This document is intended to provide a brief background on Principal Component Analysis (PCA), Singular Value Decomposition (SVD), and the Online SVD technique of Brand (Brand, 2003).

## 2 Eigenvectors and Eigenvalues

Both the Singular Value Decomposition and Principal Component Analysis are rooted in the concept of eigenvectors and eigenvalues, which are the solutions to the equation:

$$Ax = \lambda x$$

where $A \in \mathbb{R}^{n \times n}$, $x \in \mathbb{R}^{n \times 1}$, and $\lambda \in \mathbb{R}$. This equation says that for certain special vectors $x$ (the *eigenvectors*), the general transformation matrix $A$ only scales them by a factor (the *eigenvalues*) and does not rotate them. Every $n \times n$ square matrix has $n$ (not necessarily distinct) eigenvectors. They can be found by rearranging the terms of the above equation:

$$Ax - \lambda x = 0$$

$$(A - \lambda I)x = 0$$

We don't consider the 0 vector to be an eigenvector, so the above equation is true when

$$det(A - \lambda I) = 0$$

This equation is called the *characteristic polynomial*, which is a set of $n$ linear equations which can be solved for $\lambda$ to get the eigenvalues. These are then plugged back into the matrix equation to get the eigenvectors. In practice, this is only done for small matrices due to the computational complexity. For larger matrices, an iterative algorithm based on the characteristic polynomial is used.

### 2.1 Computing Eigenvectors from the Characteristic Polynomial

The characteristic polynomial can be used to develop an algorithm for computing the eigenvectors and eigenvalues. This algorithm, which is known as *Power Iteration*, takes advantage of the property that vectors which are transformed by the matrix $A$ will be scaled in the direction of the largest eigenvector. Initially, the vector is chosen to be some random non-zero vector. This vector is iteratively multiplied by the matrix, which causes it to gradually align with the largest eigenvector. This is repeated until the vector has converged. The eigenvector and eigenvalue are stored, and the eigenvector's contribution is subtracted from the matrix. The process is then repeated to compute the next largest eigenvector. Here is the pseudocode for determining an eigenvector:

```
$x_1 = \mathrm{rand}(n,1)$
$x_1 = x_1/\|x_1\|$
**repeat**
    $x_{prev} = x_1$
    $x_1 = Ax_1$
    $x_1 = x_1/\|x_1\|$
**until** $\|x_1 - x_{prev}\| > threshold$
$\lambda = \|x_1\|$
```

To compute the subsequent eigenvalues, the eigenvector is removed from $A$ in a process called *Deflation*. The outer product of the eigenvector is subtracted from the matrix:

$B = \lambda x_1 x_1^T$
$A' = A - B$

The performance of Power Iteration is highly-dependent on the eigenvalues. The convergence ratio is controlled by the ratio of subsequent eigenvalues $\frac{\lambda_i}{\lambda_{i+1}}$. If this ratio is large, the algorithm will converge rapidly. Thus Power Iteration is usually used only when the first couple of eigenvalues need to be computed, since these are usually well-seperated.

## 3 Principal Component Analysis

PCA is a widely used data compression technique... The PCA is computed by determining the eigenvectors and eigenvalues of the *covariance matrix*. The covariance of two random variables is their tendency to vary together. This expressed as:

$$\mathrm{cov}(X,Y) = E[E[X] - X] \cdot E[E[Y] - Y]$$

where $E[X]$ denotes the expected value of $X$. For sampled data this can be explicitly written out as:

$$\mathrm{cov}(X,Y) = \sum_{i=1}^{N} \frac{(x_i - \bar{x})(y_i - \bar{y})}{N}$$

with $\bar{x} = \mathrm{mean}(X)$ and $\bar{y} = \mathrm{mean}(Y)$. Note that $\mathrm{cov}(X,X) = \mathrm{var}(X)$, and for independent variables $\mathrm{cov}(X,Y) = 0$. The covariance matrix is a matrix $A$ with elements $A_{i,j} = \mathrm{cov}(i,j)$. The covariance matrix is square and symmetric. For independent variables, the covariance matrix will be a diagonal matrix with the variances along the diagonal.

To calculate the covariance matrix from a dataset, first center the data by subtracting the mean of each sample vector. Considering the columns of the data matrix $A$ as the sample vectors, we can write the elements of the covariance matrix $C$ as:

$$c_{ij} = \frac{1}{N} \sum_{i=1}^{N} a_{ij} a_{ji}$$

written in matrix form:

$$C = \frac{1}{N} AA^T$$

Often the scale factor $1/N$ is distributed throughout the matrix and the covariance matrix is written simply as $AA^T$.

The eigenvectors of the covariance matrix are the axes of maximum variance. The PCA technique is widely used because of the fact that in many datasets, a majority of the variance of the data can be captured by a small subset of the eigenvectors. An example from a surface lightfield dataset is shown in Figure 1. Note that the magnitude of the eigenvalues falls off quickly, such that 70% of the value is contained in the first 15 eigenvectors. This implies that a good approximation of the full matrix can be computed using only a subset of the eigenvectors and eigenvalues. To approximate the matrix, the eigenvalues are truncated below some threshold. The data is then reprojected onto the remaining $r$ eigenvectors to get a *rank-r* approximation.

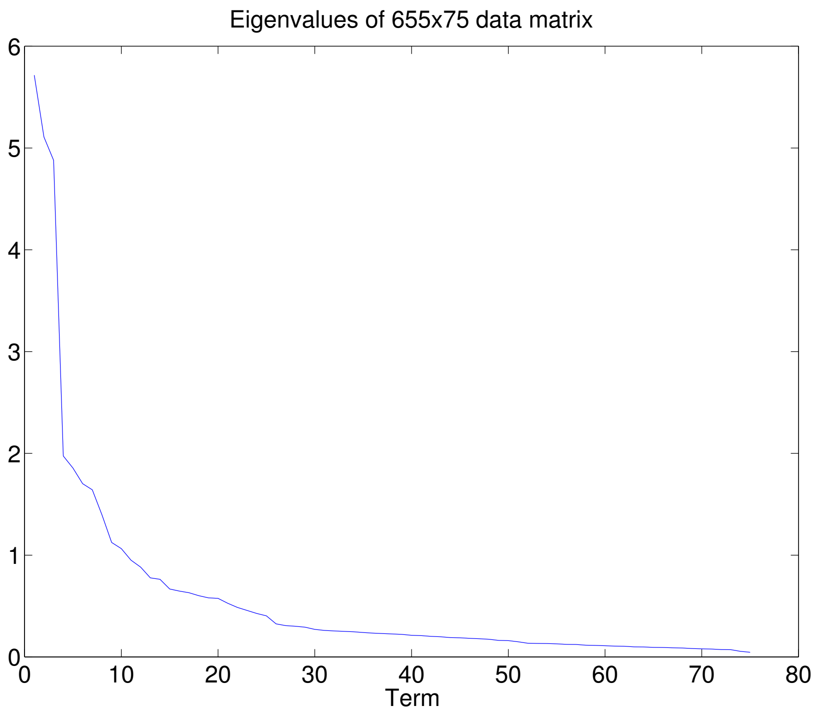

Figure 1: The eigenvalues of a 655x75 matrix from a surface lightfield dataset. The mass of the eigenvalues are concentrated in the first dozen terms, which implies that a good approximation of the full matrix can be computed using only a subset of the eigenvectors and eigenvalues.

This can be shown to be optimal compression in the least-squares sense, i.e. it is the closest approximation to the full matrix that can be achieved $r$ terms.

## 4 Singular Value Decomposition

The Singular Value Decomposition (SVD) is a technique for decomposing a matrix into a set of rotation and a scale. The form is:

$$A = USV^T$$

where $A \in \mathbb{R}^{m \times n}$ (with $m >= n$), $U \in \mathbb{R}^{m \times n}$, $V \in \mathbb{R}^{n \times n}$, and $S$ is a diagonal matrix of size $\mathbb{R}^{n \times n}$. Both $U$ and $V$ are orthogonal.

The SVD is closely related to PCA and to eigenvalue computation. Recall from above the eigenvalue equation:

$$Ax = \lambda x$$

In order to compute the eigenvalues, $A$ must be a square matrix. The SVD is less restrictive in that it can be performed on any $m \times n$ matrix. The singular values of a matrix $A$ solve the equations:

$$Au = \lambda v \text{ and } A^T v = \lambda u$$

. The vectors $u$ and $v$ are known as the right- and left-singular vectors respectively. We can show the relation between SVD and eigenvalues through the following equations:

$$AA^T = (USV^T)(USV^T)^T = USV^TVSU^T = US^2U^T$$

$$A^TA = (USV^T)^T(USV^T) = VSU^TUSV^T = VS^2V^T$$

using the fact the $U$ and $V$ are orthogonal so $U^T = U^{-1}$. This shows that $U$ and $V$ can be calculated as the eigenvectors of $AA^T$ and $A^TA$ respectively. The square root of the eigenvalues are the singular values along the diagonal of the $S$ matrix.

The advantage of the SVD is that there are a number of algorithms for computing the SVD. One method is to compute $V$ and $S$ by diagonalizing $A^TA$:

$$A^TA = VS^2V^T$$

and then to calculate $U$ as:

$$U = AVS^{-1}$$

There are also a number of efficient algorithms for computing which minimize error propagation discussed in (Golub and Loan, 1996). [*note: Discuss Lanczos? QR?*]

### 4.1 Online SVD

Singular Value Decomposition is a powerful and widely-used compression technique, but it requires that the full set of data be available during processing. Since this data is usually extensively resampled to fit the columns and rows of the data matrices, this can be a significant storage cost.

The Online Singular Value Decomposition (Brand, 2003) is an incremental SVD algorithm (Hall et al., 2000; Chandrasekaren et al., 1997) that computes the principal eigenvectors of a matrix without storing the entire matrix in memory. The results are built up from a series of simple operations on the output eigenvectors, which are low-rank approximations to the full matrix. If the rank $r$ is much smaller than the size of the matrices, this is a considerable saving, and reduces the computational complexity from quadratic to linear (Brand, 2003).

The Online SVD works as follows (see Figure 2). Consider a rank-r SVD

$$A = USV^T$$

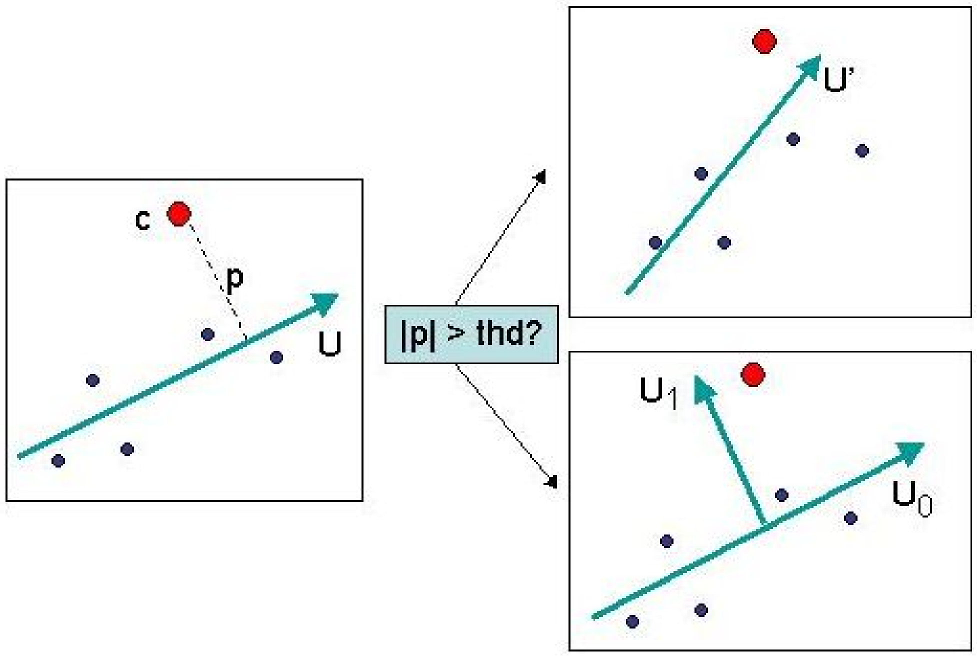

Figure 2: The Online SVD. A new point $c$ is incorporated into the existing SVD. The orthogonal component $p$ is computed by projecting onto $U$. If $\|p\|$ is below a threshold, then we can incorporate this new sample by simply rotating $U$. Otherwise, we must increase the rank of the approximation.

where $U \in \mathbb{R}^{m \times r}$, $S \in \mathbb{R}^{r \times r}$, and $V \in \mathbb{R}^{n \times r}$. As each new image is captured, it is resampled into a per-vertex column vector, which represents every pixel in a surface patch from one camera view. This column of samples $c$ is projected onto the eigenspace:

$$j = U^T c$$

The amount that is orthogonal to the eigenspace is given by:

$$p = c - Uj$$

The norm of this vector, $\|p\|$, is a measure of how different the pixels in this new image are from our current approximation. If the pixels are similar (that is, $\|p\|$ is below a threshold) we can incorporate this new sample by simply rotating the existing eigenspaces.

$$U' = UR_U \qquad V' = VR_V$$

Otherwise, the current rank $r$ of the approximation is insufficient, and we increase the rank to $r+1$ and append the column $j$ to our approximation.

$$U' = [U; j]\, R_U \qquad V' = VR_V$$

The rotations are computed by re-diagonalizing the $(r+1) \times (r+1)$ matrix

$$\begin{bmatrix} S & U^T c \\ 0 & \|p\| \end{bmatrix} \rightarrow [R_U, R_V] \tag{1}$$

These rotations can be computed in $O(r^2)$. Since only the output matrices $U$, $S$, and $V$ are stored, this representation results in significant storage savings.

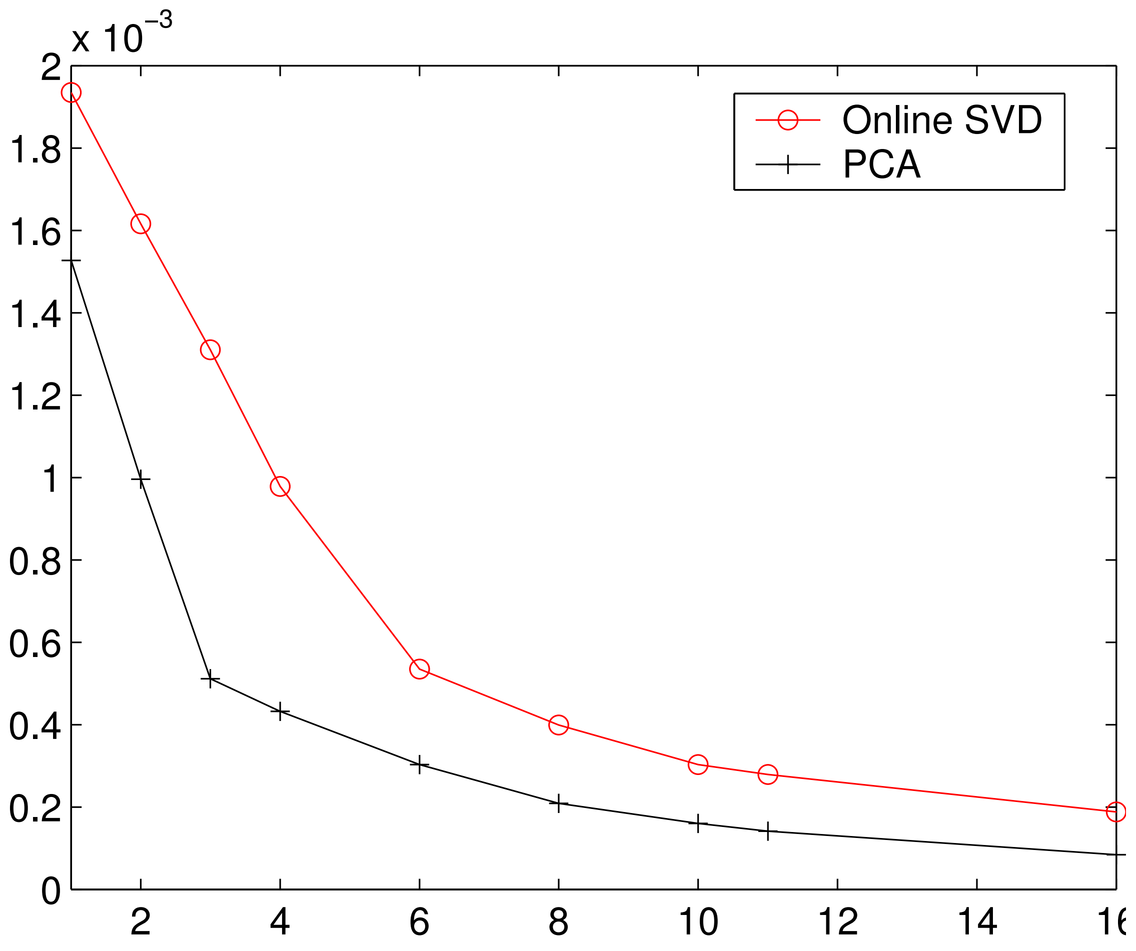

Figure 3: The Mean Squared Error of a reconstructed light field as a function of the rank. The majority of the light field is captured after 4-6 terms. For the same number of terms, the Online SVD has more error than the PCA.

#### 4.1.1 Error

Since the Online SVD is constructed incrementally, it tends to have more error than the SVD for the same rank. Figure 3 shows the error as a function of rank. A rank that is too low biases the Online SVD computation by forcing it to select sub-optimal eigenvectors. Brand (Brand, 2003) suggests computing the Online SVD at twice the actual rank and only using the largest eigenvalues. This allows the eigenvectors more degrees of freedom to fit to the incoming data.

### 4.2 Missing Values

Acquired radiance data is often incomplete because of occlusion. Many systems are forced to discard surface patches with missing data, or fill in the holes with incorrect values, such as zeros or mean values. A better approach is to estimate the missing data using a process known as *imputation*. Imputation uses the current Online SVD estimate of the light field to fill in missing values (Brand, 2003). The known samples are projected onto the current eigenspace, and the unknown values are estimated by solving the under-determined system using Linear Least Squares (Golub and Loan, 1996). This fills in the missing values with the nearest plausible values using the Mahalonobis metric (a metric defined in the scaled eigenspace) (Brand, 2003).

Suppose that an incoming data column is partitioned into two components, $\hat{c}$ and $\breve{c}$, which represent the known and unknown components of the incoming image. For example, $\breve{c}$ could represent the occluded region of a triangle. The current eigenspace is also partitioned into $\hat{U}$ and $\breve{U}$. We can set up two linear systems:

$$\begin{aligned} \hat{U}Sb &= \hat{c} \\ \breve{U}Sb &= \breve{c} \end{aligned}$$

These equations state that a vector $b$ can be projected on the known subspace and the unknown subspace to get the known column vector and the unknown column vector. We can solve these equations using the method of Normal Equations (in (Golub and Loan, 1996) and discussed in Section **??**). We solve the first equation using Normal Equations:

$$b = (S\hat{U}^T\hat{U}S)^{-1}(S\hat{U}^T\hat{c})$$

This can be simplified using the fact that $U$ is orthogonal:

$$b = (\hat{U}S)^{-1}\hat{c}$$

and plug in $\breve{U}Sk = \breve{c}$

$$\breve{c} = \breve{U}S(\hat{U}S)^{-1}\hat{c}$$

This set of equations yields the full vector $c$ which lies closest to the existing eigenspace in the Least Squares sense (for further details, consult (Brand, 2003). The inverse is not a true inverse, but is computed using the *pseudo-inverse* (discussed in Section **??**). By plugging the above equations into Equation 1, we get:

$$\begin{bmatrix} S & U^Tc \\ 0 & \|k\| \end{bmatrix} = \begin{bmatrix} S & S(\hat{U}S)^{-1}\hat{c} \\ 0 & \|\hat{c} - \hat{U}S(\hat{U}S)^{-1}\hat{c})\| \end{bmatrix}$$

Figure **??** shows the advantage of imputation for surface light fields. In practice, we only impute missing values when at least half of the surface patch is visible in an image. In addition, we only impute values after 8-10 initial images, to allow the system to establish a reasonable approximation.

### 4.3 Representing SLFs as SVDs

Chen et al. (Chen et al., 2002) use Principal Component Analysis (Golub and Loan, 1996) to extract the low-dimensional functions $g(s,t)$ and $h(\theta,\phi)$ from the full data matrices. The Online enables us to incrementally build a compressed representation of a surface light field.

## References

Brand, M. (2003). Fast online svd revisions for lightweight recommender systems. In *SIAM International Conference on Data Mining.*

Chandrasekaren, S., Manjunath, B., Wang, Y., Winkler, J., and Zhang, H. (1997). An eigenspace update algorithm for image analysis. In *Graphical Models and Image Processing.*

Chen, W.-C., Bouguet, J.-Y., Chu, M., and Grzeszczuk, R. (2002). Light field mapping: Efficient representation and hardware rendering of surface light fields. In *SIGGRAPH.*

Golub, G. H. and Loan, C. F. V. (1996). *Matrix Computations.* Johns Hopkins University Press, 3rd edition.

Hall, P., Marshall, D., and Martin, R. (2000). Merging and splitting eigenspace models. In *IEEE Trans. on Pattern Analysis and Machine Intelligence.*