A Best Basis

The problem we want to consider is this: Suppose you're given some data that is expressed as p points in $\mathbb{R}^n$. If we suppose that the data actually lies on a low dimensional subspace of $\mathbb{R}^n$, is it possible to find that low dimensional basis?

First, let's look at an extreme example- Just three points, $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}$, each in $\mathbb{R}^{4000}$. If this is all our data, and we are able to store any basis we like, then a much more efficient basis would be the three vectors. That is,

$$\mathbf{x}^{(1)} = 1 \cdot \mathbf{x}^{(1)} + 0 \cdot \mathbf{x}^{(2)} + 0 \cdot \mathbf{x}^{(3)}$$

so that $\mathbf{x}^{(1)}$ would simply be stored as $(1, 0, 0)$. Similarly, $\mathbf{x}^{(2)}$ could be stored as $(0, 1, 0)$, and the last vector as $(0, 0, 1)$. In this example, we are trading the 4000 dimensional space for the lower, 3-dimensional space.

In general, we will assume that there is a low dimensional subspace that encapsulates most of the data (rarely will data lie completely on a low dimensional subspace without some error). Our goal is to find a basis for that subspace.

There are a couple of things that will make our job easier:

- We will assume that the data has been mean-subtracted, so that the mean is zero.
- The basis is orthonormal.
- To find the "best" basis will require an error function. We will then minimize it.

At the end of this section, you'll see that the best $k-$dimensional basis for your data (regardless of k) is given by the first k eigenvectors of the covariance matrix, which are typically computed using the Singular Value Decomposition (SVD).

The Covariance Matrix

As we recall, the covariance matrix for data in $\mathbb{R}^n$ measures the covariance between the data in coordinate i and the data in coordinate j.

If we have p vectors in $\mathbb{R}^n$ organized as usual in columns, $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(p)}$, then the k^{th} coordinate of data point i will be denoted as

$$x_k^{(i)}$$

and if we assume that the mean in each coordinate is zero,

$$\frac{1}{p} \sum_{k=1}^{p} x_k^{(i)} = 0$$

then the covariance between coordinate i and coordinate j will be

$$C_{ij} = \frac{1}{p-1} \sum_{n=1}^{p} x_i^{(n)} x_j^{(n)}$$

Now, $x_i^{(n)}$ is the i^{th} "row" of $\mathbf{x}^{(n)}$ and $x_j^{(n)}$ is the j^{th} "column" of $\mathbf{x}^{(n)}$, so the entire covariance matrix can be expressed as:

$$C = \frac{1}{p-1}\sum_{p=1}^{n} \mathbf{x}^{(n)} \left(\mathbf{x}^{(n)}\right)^T$$

If the matrix X stores the data in columns so that X is $n \times p$, we can write:

$$C = \frac{1}{p-1} X X^T$$

If the matrix X stores the data in rows so that X is $p \times n$, we can express this as

$$C = \frac{1}{p-1} X^T X$$

In either event, the covariance matrix is $n \times n$.

Finally, we recognize that the covariance matrix is symmetric, so the Spectral Theorem applies. In particular, there is an orthonormal matrix P and a diagonal matrix D so that

$$C = PDP^T$$

where the columns of P form the eigenvectors associated with the diagonal elements of D (which are typically written largest to smallest).

To connect this with the SVD of the data matrix X, if X is $n \times p$ (so that data is stored column-wise), and we write the **reduced** SVD as:

$$X = U\Sigma V^T$$

Then

$$C = \frac{1}{p-1} X X^T = \frac{1}{p-1} U\Sigma V^T V \Sigma^T U^T = U\left(\frac{1}{p-1}\Sigma^2\right) U^T$$

We see a relationship between the singular values of X, σ_i, and the eigenvalues of the covariance matrix, $\hat{\lambda}_i$:

$$\frac{1}{p-1}\sigma_i^2 = \hat{\lambda}_i$$

And, of course the most important conclusion: The eigenvectors of the covariance matrix are the left singular vectors of the data matrix (when the data is written column-wise and has been mean subtracted).

Suppose that the matrix X is $3,000,000 \times 200$. Then the eigenvectors of the covariance matrix are in $\mathbb{R}^{3,000,000}$, and the matrix P (or matrix U) would be a huge matrix to have to compute.

There is a better way. Suppose that we know the **right** singular vectors in the matrix V (which reside in $\mathbb{R}^{200}$). Is there a fast way to compute a few of the columns in U from that? Yes! Recall that this is how we originally defined the vectors in U:

$$\mathbf{u}_i = \frac{1}{\sigma_i} X \mathbf{v}_i$$

Or, more practically, compute $X\mathbf{v}_i$ and divide it by its norm.

Projections and the Mean

We now look at something a little different, but we'll tie it together later. Suppose you have your p data points in $\mathbb{R}^n$ that have *not* been mean subtracted, and you have a vector $\mathbf{u}$ onto which we want to project the data.

First, if we project one point $\mathbf{x}$ onto our (unit) vector $\mathbf{u}$, then the point becomes the scalar $\mathbf{x}^T\mathbf{u}$. Similarly, projecting all the data gives us p real numbers:

$$\mathbf{u}^T\mathbf{x}^{(1)}, \mathbf{u}^T\mathbf{x}^{(2)}, \ldots, \mathbf{u}^T\mathbf{x}^{(p)}$$

so the mean of the projected data is given by

$$\frac{1}{p}\sum_{n=1}^{p}\mathbf{u}^T\mathbf{x}^{(n)} = \mathbf{u}^T\left(\frac{1}{p}\sum_{n=1}^{p}\mathbf{x}^{(n)}\right) = \mathbf{u}^T\bar{\mathbf{x}}.$$

Therefore, the mean of the projection is the projection of the mean. In particular, if the mean of a data set is zero, and the data is projected to a vector (or subspace), then the new mean is also zero.

Projections and the Basis

In this section, let's think about projecting the data to various one dimensional subspaces and how that effects the variance of the data. We'll keep our previous general data set, p points in $\mathbb{R}^n$, and we'll suppose that the data has been mean subtracted (although we could do the mean subtraction after the projection, as we showed in the previous section).

If we project the data onto an arbitrary unit vector $\mathbf{u}$, then the resulting variance (as scalars) will be:

$$\begin{aligned}
S_{\vec{u}}^2 &= \frac{1}{p-1}\sum_{n=1}^{p}(\mathbf{u}^T\mathbf{x}^{(n)})^2 \\
&= \frac{1}{p-1}\sum_{n=1}^{p}\mathbf{u}^T\mathbf{x}^{(n)}(\mathbf{x}^{(n)})^T\mathbf{u} \\
&= \mathbf{u}^T\left(\frac{1}{p-1}\sum_{n=1}^{p}\mathbf{x}^{(n)}(\mathbf{x}^{(n)})^T\right)\mathbf{u} = \mathbf{u}^T C\mathbf{u}
\end{aligned}$$

This is actually a very key quantity, and will come up in the next section. We will look at this quantity more closely in a bit, but let's look at what happens in one special case: Suppose that $\mathbf{u}$ is an eigenvector of the covariance C corresponding to the first eigenvalue of C, $\hat{\lambda}_i$ using our previous notation. Then:

$$\mathbf{v}_1^T C\mathbf{v}_1 = \mathbf{v}_1^T\hat{\lambda}_1\mathbf{v}_1 = \hat{\lambda}_1$$

Therefore, if we project all the data to the first eigenvector of C, the new variance will be the first eigenvalue of C.

Reconstruction Error and the Basis

Given a specific vector $\mathbf{x} \in \mathbb{R}^n$ and an arbitrary orthonormal basis, $\vec{\phi}_1, \vec{\phi}_2, \ldots, \vec{\phi}_n$, we can write

$$\mathbf{x} = (\vec{\phi}_1^T \mathbf{x})\vec{\phi}_1 + (\vec{\phi}_2^T \mathbf{x})\vec{\phi}_2 + \ldots + (\vec{\phi}_n^T \mathbf{x})\vec{\phi}_n$$

so that the magnitude of $\mathbf{x}$ can be written as:

$$\|\mathbf{x}\|^2 = (\vec{\phi}_1^T \mathbf{x})^2 + (\vec{\phi}_2^T \mathbf{x})^2 + \ldots + (\vec{\phi}_n^T \mathbf{x})^2.$$

We can use the same algebraic manipulation that we used in the last section to rewrite this as:

$$\|\mathbf{x}\|^2 = \vec{\phi}_1^T \mathbf{x}\mathbf{x}^T \vec{\phi}_1 + \vec{\phi}_2^T \mathbf{x}\mathbf{x}^T \vec{\phi}_2 + \ldots + \vec{\phi}_n^T \mathbf{x}\mathbf{x}^T \vec{\phi}_n.$$

We can break this up and define the error using one vector $\vec{\phi}_1$:

$$\|\mathbf{x}\|^2 = \vec{\phi}_1^T \mathbf{x}\mathbf{x}^T \vec{\phi}_1 + \|\mathbf{x}_{\text{err}}\|^2$$

Now do this for all p data points. For any single vector $\vec{\phi}_1$, we sum these together:

$$\begin{aligned}
\|\mathbf{x}^{(1)}\|^2 &= \vec{\phi}_1^T \mathbf{x}^{(1)}(\mathbf{x}^{(1)})^T \vec{\phi}_1 + \|\mathbf{x}_{\text{err}}^{(1)}\|^2 \\
\|\mathbf{x}^{(2)}\|^2 &= \vec{\phi}_1^T \mathbf{x}^{(2)}(\mathbf{x}^{(2)})^T \vec{\phi}_1 + \|\mathbf{x}_{\text{err}}^{(2)}\|^2 \\
&\vdots\vdots \\
\|\mathbf{x}^{(p)}\|^2 &= \vec{\phi}_1^T \mathbf{x}^{(p)}(\mathbf{x}^{(p)})^T \vec{\phi}_1 + \|\mathbf{x}_{\text{err}}^{(p)}\|^2
\end{aligned}$$

$$\sum_{n=1}^{p} \|\mathbf{x}^{(n)}\|^2 = \vec{\phi}_1^T \left(\sum_{n=1}^{p} \mathbf{x}^{(n)}(\mathbf{x}^{(n)})^T \right) \vec{\phi}_1 + \sum_{n=1}^{p} \|\mathbf{x}_{\text{err}}^{(n)}\|^2$$

We can multiply everything by $1/p$ or $1/(p-1)$ to make things work. That is,

$$\frac{1}{p-1} \sum_{n=1}^{p} \|\mathbf{x}^{(n)}\|^2 = \vec{\phi}_1^T C \vec{\phi}_1 + \frac{1}{p-1} \sum_{n=1}^{p} \|\mathbf{x}_{\text{err}}^{(n)}\|^2. \qquad (1)$$

In light of this equation, let us now define an error function using an arbitrary orthonormal basis, $\vec{\phi}_1, \ldots, \vec{\phi}_n$. The error we get when using a one dimensional representation of our data is given by

$$E(\vec{\phi}_2, \ldots, \vec{\phi}_n) = \frac{1}{p-1} \sum_{n=1}^{p} \|\mathbf{x}_{\text{err}}^{(n)}\|^2.$$

Notice that the left side of Equation 1 is constant. Therefore, minimizing our error function is equivalent to maximizing $\vec{\phi}_1^T C \vec{\phi}_1$.

Here now is our algorithm for find the "best" basis:

1. Find the unit vector $\vec{\phi}_1$ so that $\vec{\phi}_1^T C \vec{\phi}_1$ is maximized.
2. We "project out" this vector so that the i^{th} data point now becomes:
$$\underline{\mathbf{x}}^{(i)} = \mathbf{x}^{(i)} - \text{Proj}_{\vec{\phi}_1}(\mathbf{x}^{(i)})$$
3. Re-compute C.
4. Repeat from Step 1 until we have enough basis vectors.

In practice, we will not need to do this- there is an easier way!

The Best Basis and the Eigenvectors

We have shown that finding the best basis reduces to maximizing the quantity:

$$\max_{\vec{\phi} \neq \vec{0}} \frac{\vec{\phi}^T C \vec{\phi}}{\vec{\phi}^T \vec{\phi}}$$

where we divide by the magnitude (squared) to enforce the fact that we want a unit vector, and we want to stay away from the zero vector.

We know that the eigenvectors of the covariance matrix form an orthonormal basis for $\mathbb{R}^n$, so we can write any vector as a linear combination of them:

$$\vec{\phi} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n = V\mathbf{c}$$

Using the eigenvector-eigenvalue factorization $C = VDV^T$, we can now write the numerator as:

$$\vec{\phi}^T C \vec{\phi} = (V\mathbf{c})^T(VDV^T)(V\mathbf{c}) = \mathbf{c}^T(V^TV)D(V^TV)\mathbf{c} = \mathbf{c}^TD\mathbf{c} = c_1^2\lambda_1 + c_2^2\lambda_2 + \cdots + c_n^2\lambda_n$$

Similarly, the denominator is:

$$\vec{\phi}^T\vec{\phi} = c_1^2 + c_2^2 + \cdots + c_n^2$$

Let's look at the coefficients of our expansion now. For λ_i, the coefficient in front is

$$\rho_i = \frac{c_i^2}{c_1^2 + c_2^2 + \cdots + c_n^2}$$

where $\rho_i \geq 0$ and $\sum_{i=1}^n \rho_i = 1$ (like a probability distribution). Let's summarize where we are. We now see that maximizing $\vec{\phi}^T C \vec{\phi}$ is equivalent to choosing $\rho_1, \rho_2, \cdots, \rho_n$ so that each $\rho_i \geq 0$ and they sum to 1, to maximize the quantity:

$$\rho_1\lambda_1 + \rho_2\lambda_2 + \cdots + \rho_n\lambda_n$$

It is easy to see that, if the $\lambda_i \geq 0$ and are ordered from largest to smallest, then:

$$\lambda_n \leq \rho_1\lambda_1 + \rho_2\lambda_2 + \cdots + \rho_n\lambda_n \leq \lambda_1.$$

To maximize our given quantity, we set $c_1 = 0$ and the rest of the coefficients to zero. This leads us to our main conclusion:

The Best Basis The vector $\vec{\phi}$ that maximizes the quantity

$$\max_{\vec{\phi}\neq\vec{0}} \frac{\vec{\phi}^T C \vec{\phi}}{\vec{\phi}^T \vec{\phi}}$$

is given by $\mathbf{v}_1$, the eigenvector corresponding to the largest eigenvalue of C. Therefore, the best $k-$dimensional basis for a given set of data is found by taking the first k eigenvectors of the covariance matrix. We note in passing that the value of the maximum would then be the first eigenvalue of the covariance matrix.

Alternatively, given the data in an $n \times p$ matrix X, the best $k-$dimensional basis is found by taking the first k columns of the U, the left singular vectors of the SVD of X.

The Best Basis and The Rank

Before we start the decomposition, it would be a good idea to decide on how the rank of X will be computed- Matlab uses a very very small tolerance on its rank cutoff, so it will often return the maximum rank.

It is useful to look at the rank as that number of basis vectors required to preserve some percentage of the variance in the data. From our previous section on the covariance matrix, we had a relationship between the eigenvalues of the covariance matrix, $\hat{\lambda}_i$ and the singular values of X:

$$\frac{1}{p-1}\sigma_i^2 = \hat{\lambda}_i$$

so normalizing the set of eigenvalues is equivalent to doing it to the squared singular values:

$$\lambda_i = \frac{\hat{\lambda}_i}{\sum_{j=1}^n \hat{\lambda}_j} = \frac{\frac{1}{p-1}\sigma_i^2}{\sum_{j=1}^n \frac{1}{p-1}\sigma_j^2} = \frac{\sigma_i^2}{\sum_{j=1}^n \sigma_j^2}$$

Now the λ_i are positive and sum to 1. How many dimensions we keep is usually problem dependent. There are some problems where keeping 60% of the variance is more than enough, and some problems will come up where we need 99% of the variance.

Best Basis and Matlab

In this section, we look at the Matlab commands that we use in finding and analyzing the best basis.

Suppose we have p points in $\mathbb{R}^n$, and these are stored in an $n \times p$ matrix X. First, we mean-subtract the data (as vectors in $\mathbb{R}^n$). I will call the mean subtracted data $\hat{X}$. Once we mean subtract, find the SVD.

```
[n,p]=size(X);
Xmean=mean(X,2);
Xhat=X-repmat(Xmean,1,p);
[U,S,V]=svd(Xhat,'econ');
temp=diag(S).^2;
Lambda=temp/sum(temp);   %Lambda contains the normalized evals
```

Now we can plot the data in `Lambda` to see what the eigenvalues look like:

```
plot(Lambda);
```

If there is a break in the values where suddenly all the remaining eigenvalues are zero, then it is relatively easy to determine the rank.

Suppose we want to find and plot the best two dimensional representation to the data in X. This low dimensional representation is provided by the coordinates, and we find them using matrix notation:

```
Xcoords=U(:,1:2)'*Xhat;  %Size of Xcoords is now 2 x p (p points in R^2)
plot(Xcoords(1,:),Xcoords(2,:),'.');
```

To put the data back into $\mathbb{R}^n$ (as a 2 or k dimensional approximation), we would just multiply by the appropriate columns of U:

```
Xrecon=U(:,1:2)*Xcoords;  %Project the data back to R^n
```

An Example in Detail

Download the data `author.mat` from the class website. The data represents a short movie[1] consisting of 109 frames, where each frame is 120×160, or 19200 pixels. Thus, loading the data will give you an array that is 19200×109 (this is what we were referring to as $n \times p$ in the notes).

Also, we should note that the data in matrix, saved as `Y1`, consists of integer values between 0 and 255 (representing 255 shades of gray from black (0) to white (255)). This is important to mention: we'll be translating these images into vectors of real numbers instead of integers, which may impact how the coloring takes place. Discussing this in detail now would take us too far away from the discussion, but is something to keep in mind.

[1]The movie is actually a short sequence I took from my webcam when I was on sabbatical and needed some data quick!

Visualizing a vector as an array

When visualizing these large scale vectors, remember that any vector in $\mathbb{R}^{19200}$ could be visualized as a 120×160 array. In Matlab, this is performed by using the "reshape" command. We use `imagesc`, short for "image scaled", where Matlab will scale the values in the array to ready it for the color map. To see how the color mapping is performed, you can use the `colorbar` command. In this case, if $\vec{u}$ is a vector in $\mathbb{R}^{19200}$, then we can reshape it into an array and visualize it as an array using a gray scale color scheme by issuing the following commands, issued on one line

```
imagesc(reshape(u,120,160)); colormap(gray); axis equal; colorbar
```

To watch the "movie", you can use the following code, which just goes through the data column by column and shows each as frame. I won't use the color bar, and the pause command is there so we can actually see the result (otherwise, the frames go by way too fast!).

```
for j=1:109
  A=reshape(Y1(:,j),120,160);
  imagesc(A); colormap(gray); axis equal; axis off;
pause(0.1)
end
```

A few sample frames and the movie's mean frame are shown in Figure 1. You might find it interesting that that the mean is the background. Once we've found the mean, then we can proceed with the Singular Value Decomposition. Now consider the following code "snippet":

```
X=Y1-repmat(Ymean,1,109);
[U,S,V]=svd(X,'econ');
tempS=diag(S).^2;
tempS=tempS/sum(tempS)
```

The last line stores the normalized eigenvalues (singular values squared) into the vector `tempS`. Now let's consider the rank. Here are the first few of the values in that vector:

$$0.3436 \qquad 0.1229 \qquad 0.0847 \qquad 0.0566 \qquad 0.0407$$

There seems to be a natural break between the first two and the third. Keeping the data two dimensional would retain approximately $34.36 + 12.29 = 46.6\%$ of the variance in the data. Keeping three dimensions takes us to about 55%. We need to go to 22 dimensions to retain 90%.

In the script file, we'll compare a two dimensional reconstruction, a 10 dimensional reconstruction and a 22 dimensional reconstruction of the data. This is done by using two, 10, then 22 basis vectors for the subspace in which the data lies.

First, let's examine the actual basis vectors. From our code snippet, the basis vectors of interest are the columns of U (the left singular vectors of X). Since each basis vector is in $\mathbb{R}^{19200}$, the basis vector(s) can also be visualized as an array.

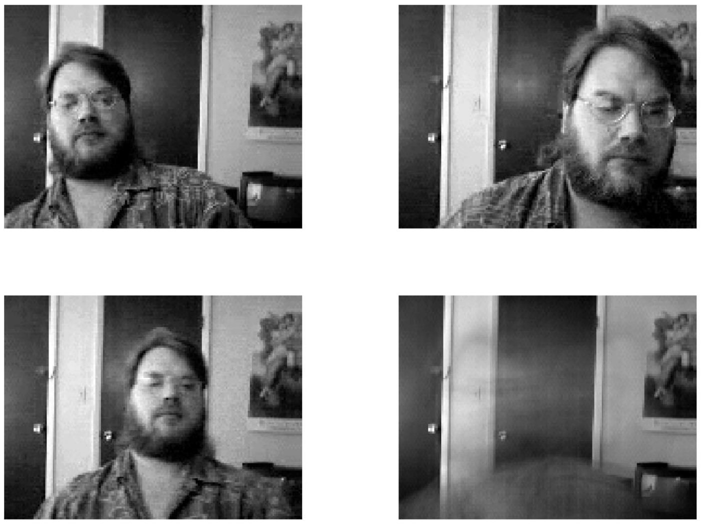

Figure 1: The figure above has three sample frames (Frames 23, 60 and 80) and the movie's mean.

Sample Code

```
%% Sample script for the "author" data

% Some cleaning commands:
close all;  %Close all open figures
clear       %Clear the memory
clc         %Clear the command window

% Download the data from the class website, so it can be loaded here:
load author
Y1=double(Y1);  %Changes the data from "integers" to "floating point"
                %The data was saved as integers to save on memory.

% We can visualize the "movie"
figure(1)
for j=1:109
   A=reshape(Y1(:,j),120,160);
   imagesc(A); colormap(gray); axis equal; axis off;
   pause(0.1)
end

%% Mean subtraction
Ymean=mean(Y1,2);
```

```
figure(2)
imagesc(reshape(Ymean,120,160)); colormap(gray); axis equal; axis off;
title('The Movie Mean');

%% The SVD
% We'll mean subtract, but keep the original data intact.  We'll call put
% the mean subtracted data in matrix X.
X=Y1-repmat(Ymean,1,109);
[U,S,V]=svd(X,'econ');
tempS=diag(S).^2;
tempS=tempS/sum(tempS);   %Normalized eigenvalues.

figure(3)
plot(tempS);
title('The Eigenvalues of X^TX (Singular Values squared)')

%% Projection to R^2
Xcoords=U(:,1:2)'*X;   %The matrix Xcoords is now 2 x 109
figure(4)
plot(Xcoords(1,:),Xcoords(2,:),'.');
title('The movie data presented in the best two dimensional basis');

%% The Reconstruction back into 19200 dim space
Xrecon=U(:,1:2)*Xcoords;  %Add back the mean if needed.
```

Exercises

1. Suppose we have p data points in $\mathbb{R}^n$. Show that the variance of the data, projected to a standard basis vector $\vec{e}_i$, returns the standard variance in that dimension.

2. By hand, be able to reproduce the computation that shows that the variance of the scalar projection of the p data points in $\mathbb{R}^n$ to a unit vector $\mathbf{u}$ is given by

$$\mathbf{u}^T C \mathbf{u}$$

3. Suppose we have two o.n. vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$. If we still have our p points in $\mathbb{R}^n$, compute a quantity similar to the last exercise that gives the covariance between the data projected to $\mathbf{u}$ and the data projected to $\mathbf{v}$.

   In the special case that $\mathbf{u}, \mathbf{v}$ are eigenvectors of the covariance matrix, how does this quantity simplify?

4. Relate the variance to the error. That is, be able to reproduce the argument that

$$\sum_{n=1}^{p} \|\mathbf{x}^{(n)}\|^2 = \vec{\phi}_1^T \left(\sum_{n=1}^{p} \mathbf{x}^{(n)} (\mathbf{x}^{(n)})^T \right) \vec{\phi}_1 + \sum_{n=1}^{p} \|\mathbf{x}_{\text{err}}^{(n)}\|^2$$

5. In Matlab, run the sample code for `author.mat`, and answer the following:
   (a) Verify numerically that the variance of the projected data to the best basis vector (first one) is given by the first eigenvalue of the covariance matrix.
   (b) Verify numerically that we have a good basis in using the first two eigenvectors of the covariance matrix (the first two columns of U in the SVD) by creating 500 random orthonormal pairs of vectors and comparing the reconstruction errors. Here's a start for your code:

```
%Assume X is given as in the previous "author" code.
for j=1:500
    [Q,temp]=qr(randn(19200,2),0);  %Q that has o.n. cols, discard temp
    Error=????  (A matrix whose columns are the x_err)
    ErrJ=norm(Error,'fro');
end
```

Project: Eigenfaces

Suppose we have a large collection of photos of people's faces, and that each photo is $u \times v$. Then we think of each "face" as a vector in $\mathbb{R}^n$ (where $n = uv$).

If we look for a "best basis" in this space, we will get basis vectors that are also in $\mathbb{R}^n$, and these can be viewed as an array of $u \times v$ pixels. Therefore, if the basis vectors are the eigenvectors of the covariance matrix, we might call such vectors "eigenfaces".

We already know how to compute such a basis. There is data on our class website where each face has $262 \times 294 = 77,028$ pixels.

Our project is to:

1. Compute the mean vector and represent it as a face.
2. Plot a sample of 5 faces and the mean face together in one plot.
3. Compute the first six eigenfaces (and represent them as faces). Put them together in one plot.
4. What might be a good approximation to the rank of the matrix (if our goal is to make the faces recognizable, about 74% of the variance).
5. Plot the data using the best two dimensional representation using a new figure. Plot the figures representing females as asterisks and the figures representing males as triangles. Do you see anything interesting?