SUMMARY: Find the Best Basis

1. PREPROCESSING:

   We want to translate each image with $m \times n$ pixels as a vector with mn elements. It doesn't matter much how this is done, as long as you're consistent. The main idea here is that the space of images will be *isomorphic* to the space of vectors that we create. Here is a command that will take a single image and create a vector- You will want to store the size of the original image so you can translate the vector back into a picture:

   ```
   [m,n]=size(Image);
   v=Image(:);
   ```

   We will assume that the matrix X is $p \times k$ which represents k images, each is $m \times n$ ($p = mn$). We need to mean subtract so that the subspace is centered at the origin:

   ```
   [p,k]=size(X);
   mX=mean(X,2);  %mX is a vector that is mn
   A=X-repmat(mX,1,k);
   ```

   To continue, we assume that the matrix A has the mean subtracted picture, and A is $p \times k$.

2. GET A BASIS OF IMAGES:

   We are computing the matrix U from the SVD of $A = U\Sigma V^T$. Depending on the sizes of p and k, you might want to use different methods for computation. Your decision would normally depend on how much memory the computer has.

   - Method 1: Do the SVD directly: `[U,S,V]=svd(A,0);`
     This will return a matrix U that is $p \times n$, S is $n \times n$, and V is $n \times n$.
   - Method 2: Do eigenvector decompositions. If k is very small and you don't have much memory, this technique will work quite well.

     ```
     C=(1/p)*A'*A;
     [V,S1,V]=svd(C);  %I like the SVD rather than eig so that the
                       %elements of S1 are *ordered*
     U1=A*V;           %These are the columns of U- not scaled
     nU=sqrt(sum(U1.*U1));  %This is a row of column norms
     U=U1./repmat(nU,p,1);  %This makes U have normalized columns
     ```

   In each case, the matrix U was the desired result. Note that, in linear algebra notation, the coordinates of each face with respect to the basis vectors in U is given by: $U^T A$. The projection of every face into the subspace spanned by the columns of U is $UU^T A$. To properly visualize the results, we'll also want to add the mean back in. It might also be interesting to look at the "error"- $A - UU^T A$.

3. POSTPROCESSING: Visualizing the results

   The basic command will be to make a vector back into a matrix, then visualize the matrix using Matlab. In the following command, m, n have been defined previously, v is a vector that has mn elements, and `Image1` is the matrix output:

   ```
   Image1=reshape(v,m,n);
   imagesc(Image1);
   ```

   Here's a sequence of commands that will open several figures. It uses the matrix X, A, U, vector mX, and scalars m, n as defined in (1) and (2)

```
figure(1)           %Figure 1 has the first 4 (original) faces
colormap(gray)
for i=1:4
  subplot(2,2,i);
  imagesc(reshape(X(:,i),m,n));
end

figure(2)         %Figure 2 is the "Mean Face"
colormap(gray);
imagesc(reshape(mX,m,n));

figure(3)         %Visualize the first 4 basis images
colormap(gray);
for i=1:4
  subplot(2,2,i);   %Creates an array of plots in Figure 2
  imagesc(reshape(U(:,i),m,n));
end

K=4;            %CHANGE THIS NUMBER?
                %Number of basis vectors to use for
                %  face reconstruction
Coords=U(:,1:K)'*A;   %These are the coords of each face
                      %  using the first K basis vectors
Recon=U(:,1:K)*Coords; %These are the reconstructed images
                       %  using the first K basis vectors
figure(4)       %These are the reconstructed faces
colormap(gray);
for i=1:4
  subplot(2,2,i);
  imagesc(reshape(Recon(:,i)+mX),m,n);
end

figure(5)  %Visualize the projection of the faces to
           %the plane (points are black triangles):
plot(Coords(1,:),Coords(2,:),'b^');
```