5.5 A Summary

Before we continue, let's summarize the process for PCA (or best basis), and let's discuss what you can get out of the PCA. First, the process- It's short.

Principal Components Analysis

Let X store p points, each in $\mathbb{R}^n$ so X is $n \times p$. To find the best basis (in $\mathbb{R}^n$):

1. Compute the mean so that $\bar{\boldsymbol{x}} \in \mathbb{R}^n$.
2. Mean subtract the data, put in the $n \times p$ matrix X_m.
3. For the resulting data, take the SVD:
$$X_m = U\Sigma V^T$$
4. The best basis (or principal components) are the first k columns of U.

Once this is done, what can we do with it? Here is a list of common tasks.

- Visualize the mean and eigenvectors.

  The eigenvectors of the covariance matrix (the columns of U) typically carry very interesting information- What do they look like? In the application below, the eigenvectors can be visualized as photos, but you can also simply plot an eigenvector if that has meaning (plot the vector index along the horizontal axis, and the vector values along the vertical axis. In Matlab, if $\boldsymbol{v}$ is a vector, this would be `plot(v)`.

- Determine the rank.

  We talked about this earlier- The rank of a matrix is critical to know if you'll be constructing the pseudo-inverse, or if you simply want to reduce the dimensionality of the data (see the next item). One way to determine the rank is to look for large breaks in the plot of the singular values (the diagonal elements of Σ). A second way is to construct the eigenvalues of the covariance matrix, and retain enough dimensions so that the resulting space encapsulates a certain percentage of the variance.

- Reduce the dimensionality of the data.

  This is the primary goal of most applications of PCA. If X_m is $n \times p$ (p points in $\mathbb{R}^n$, with mean subtracted), then the following is the $k \times p$, or $k-$dimensional, representation of the p points- matrix L (for low dimensional) is $k \times p$:
$$L = U(:, 1:k)^T X_m$$
  The Matlab-esque notation means to take all rows, but only the first k columns of U.

- Visualize the data that has been reduced in dimension.

  It's often a good idea to plot the first rows of L in $\mathbb{R}^2$ just to be sure it gives you what you expect. There might be even more interesting information using other basis vectors down further in the decomposition (corresponding to lower rows of L)- We'll actually see an example below.

- Project new data using the new basis.

  If D is $m \times n$ (m points in $\mathbb{R}^n$), and D_m is the matrix with the mean of X subtracted from the columns, then the projected data in matrix P is also $m \times n$ and is given by:
$$P = U(:, 1:k)U(:, 1:k)^T D_m$$
  You can then compare P to D_m (or add the mean back to both matrices) to see if the new data is well represented by the basis.

Before we leave this discussion for an application, have you considered what the matrix V (from the SVD of X) represents? It represents a scaled version of the coordinates (or low dimensional representation). How? If $X_m = U\Sigma V^T$ is the reduced SVD, then

$$L = U^T X_m = U^T(U\Sigma V^T) = \Sigma V^T$$

And remember that Σ is just a diagonal matrix ($k \times k$) and V is $p \times k$. Multiplying by the diagonal just scales each column of V, so V is a scaled version of the $k-$dimensional representation of the data!

5.6 Application: Eigenfaces

Consider the set of all photos of some fixed width and length (say $a \times b$) and these are photos of faces. To be even more specific, let's set the coordinates of the eyes to be the same in all photos.

As you might imagine, the dimension of this "face space" should be quite large, since we're representing all possible faces here. However, should it fill up $\mathbb{R}^{ab}$? If you were to take an arbitary vector in $\mathbb{R}^{ab}$ and visualize it as an $a \times b$ photo, what would you see? The image would probably be just noise- the set of "faces" is then (hopefully) only a "small" dimensional subspace of $\mathbb{R}^{ab}$, and that gives us some reason to expect that we can find a smallish number of template faces so that every face is a linear combination of the templates.

What are these template faces? They are basis vectors in $\mathbb{R}^{ab}$, meaning that each basis vector can be visualized as an $a \times b$ image. We know that these basis vectors are the eigenvectors of the covariance matrix of the photo data, so we'll call them **eigenfaces**.

In out next example, we'll explore face space a bit and see some interesting things.

5.6.1 Project Example: The Yale Database

The data we'll look at is a portion of the Extended Yale Database below linked below if you want to have a look (good as of March 2021):

Extended Yale Database

Further, this little project is adapted from Brunton and Kutz' text on "Data Driven Science and Engineering", which is at `databookuw.com`. The had a very cute example!

As a side note, it's important to distinguish between an $m \times n$ matrix (which as m rows and n columns) versus an $m \times n$ image, which has a width of m pixels and a height of n pixels (not my fault!). The images below are said to be 168×192, but are typically stored in a 192×168 matrix (for future reference, $168 \times 192 = 32256$).

Discussion of the Data

We'll begin with a discussion of the dataset `allFaces2.mat`, which will be available from our class website. Here is a discussion of the datasets you'll see stored in this file.

- `faces` is 32256×2410 matrix, representing 2410 photos, each of which is 192×168 (as a matrix).
- Constants $n = 192$ and $m = 168$.
- `nfaces` is a vector with 38 elements, each representing one of the persons posing for photos. The datain `faces` is stored in order with all the poses for the first person first, then the second person, and so on. The vector `nfaces` stores the number of poses each person has. For example, the first person has 64 poses, so `nfaces(1)` is 64.
- `Dog` is an image of the same size as the photos of the people, but is the grayscale photo of a dog.

Figure 5.1: The first 36 of the 38 persons in the file are shown to the left. To the right, we show a sample of 36 poses for the first person (out of 64 included in the data file). Each photo is 168×192 (as an image).

Visualizing a Sample of Data

In Figure 5.1, the first 36 of the 38 persons in the file are shown to the left. To the right, we show a sample of 36 poses for the first person (out of 64 included in the data file).

Project Outline

We're going to find the best basis for our face data. Since we have so many photos, we're going to try "training" the PCA on just the photos from the first 36 people, leaving the last two people out to see how well we have captured "face space". Continuing, here's what we'll do on the computer:

1. Load the data. The "training" data will be the first 2282 columns.
2. Find the mean, then mean-subtract the data. Visualize the mean.
3. Find the SVD.
4. Look at the singular values to see if there is any clear break.
5. Look at some sample eigenfaces (columns of U).
6. Project persons 2 and 7 into the plane spanned by the 5th and 6th basis vector, and plot the result.
7. Look at reconstructions of person 37 using dimension: $50, 150, 500, 1000$.
8. Look at whether or not we can reconstruct the dog's face using a basis from people!

Project Implementation: Matlab/Octave

Side Remark: This data will most likely be too large to run on Octave-online. However, if you have Octave set up on your home PC, it should work. For the homework, we'll be using a smaller data set.

```
%% Eigenface Example using Yale Database

%%Step 1: Load the data and create the training subset using 36 people.
%   The total number of columns needed is: sum(nfaces(1:36))

load allFaces2.mat
trainingFaces=faces(:,1:sum(nfaces(1:36)));

%% Step 2: Find the mean, mean-subtract the data.  Visualize the mean
%    as a photo!

avgFace=mean(trainingFaces,2);
Xm=trainingFaces-avgFace;

figure(1)
imagesc(reshape(avgFace,n,m));
colormap(gray); axis off; axis square

%% Step 3: Compute the SVD.
[U,S,V]=svd(Xm,'econ');

%% Step 4: Look at the singular values; usually a log plot for high dimensions.
plot(log(diag(S)));  % At approx 2262, we see a big drop- this is the rank.

%% Step 5:  Let's look at some eigenfaces!  Here are the first four:
figure(2)

for j=1:4
    subplot(2,2,j)
    imagesc(reshape(U(:,j),n,m));
    colormap(gray); axis off; axis square;
end

%% Step 6:  Project the poses from persons 2 and 7 into the plane spanned by
%          basis vectors 5 and 6.  Don't worry about the formulas used to determine
%          the columns.  They're included here in case you want to change 2 and 7
%          to other people.
P1=2;  P2=7;  Bas1=5; Bas2=6;
P1cols=sum(nfaces(1:P1-1))+1:sum(nfaces(1:P1));
P2cols=sum(nfaces(1:P2-1))+1:sum(nfaces(1:P2));
Data=faces(:,[P1cols,P2cols]) - avgFace;

Coords=U(:,[Bas1,Bas2])'*Data;  % This is two dimensional data
figure(3)
plot(Coords(1,1:nfaces(P1)),Coords(2,1:nfaces(P1)),'ro');
hold on
plot(Coords(1,nfaces(P1)+1:end),Coords(2,nfaces(P1)+1:end),'k^');
hold off

%% Step 6A: Optional subplot with the two faces and positive/negative
%   5th basis vector.
```

```
subplot(2,2,1)
imagesc(reshape(faces(:,65),n,m));
colormap(gray); axis off; axis square;
subplot(2,2,2)
imagesc(reshape(U(:,5),n,m));
colormap(gray); axis off; axis square;
subplot(2,2,3)
imagesc(reshape(faces(:,381),n,m));
colormap(gray); axis off; axis square;
subplot(2,2,4)
imagesc(reshape(-U(:,5),n,m));
colormap(gray); axis off; axis square;

%% Step 7: Look at partial reconstructions of person 37 using rank 50, 150, 500, 1000
Data=faces(:,sum(nfaces(1:36))+1);
Data=Data-avgFace;
kdim=[50,150,500,1000];

figure(4)
for j=1:4
    subplot(2,2,j)
    Recon=U(:,1:kdim(j))*(U(:,1:kdim(j))'*Data);
    imagesc(reshape(Recon+avgFace,n,m));
    colormap(gray); axis off; axis square
end

%% Step 8:  Repeat, for the photo of a dog!
Data=Dog;
Data=Data-avgFace;
kdim=[50,100,400,600,1600];

figure(5)
subplot(2,3,1)
imagesc(reshape(Dog,n,m));
colormap(gray); axis off; axis equal

for j=1:5
    subplot(2,3,j+1)
    Recon=U(:,1:kdim(j))*(U(:,1:kdim(j))'*Data);
    imagesc(reshape(Recon+avgFace,n,m));
    colormap(gray); axis off; axis equal
end
```

Project Implementation: Python

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.io

# Step 0: set some parameters, load the original Matlab data
plt.rcParams['figure.figsize'] = [8, 8]
plt.rcParams.update({'font.size': 18})
```

```
mat_contents = scipy.io.loadmat('allFaces2.mat')
faces = mat_contents['faces']
m = int(mat_contents['m'])
n = int(mat_contents['n'])
Dog = mat_contents['Dog']
nfaces = np.ndarray.flatten(mat_contents['nfaces'])

#%%
# Step 1: Load the data
trainingFaces = faces[:,:np.sum(nfaces[:36])]

#%%
# Step 2: Find the mean, visualize it.
avgFace = np.mean(trainingFaces,axis=1) # size n*m by 1
Xm=trainingFaces-avgFace[:,np.newaxis];

plt.figure()
plt.imshow(np.reshape(avgFace,(m,n)).T)
plt.set_cmap('gray')
plt.title('Mean Face')
plt.axis('off')
plt.show()

#%%
# Step 3: Compute the SVD
U, S, VT = np.linalg.svd(Xm,full_matrices=0)

#%%
# Step 4: Plot the singular values.
plt.figure()
plt.plot(np.log(S))

#%%
# Step 5:  Let's look at some eigenfaces (first 4, pos and neg)

fig,axs = plt.subplots(2,2)
axs=axs.ravel()
for i in range(4):
    axs[i].imshow(np.reshape(U[:,i],(m,n)).T)

fig,axs = plt.subplots(2,2)
axs=axs.ravel()
for i in range(4):
    axs[i].imshow(np.reshape(-U[:,i],(m,n)).T)
    axs[i].set_title(str(i))


#%%
## Step 6:  Project person 2 and 7 onto basis vecs 5 and 6

P1 = 2 # Person number 2
```

```
P2 = 7 # Person number 7

P1data = faces[:,np.sum(nfaces[:(P1-1)]):np.sum(nfaces[:P1])]
P2data = faces[:,np.sum(nfaces[:(P2-1)]):np.sum(nfaces[:P2])]

P1data = P1data - avgFace[:,np.newaxis]
P2data = P2data - avgFace[:,np.newaxis]

# Project onto PCA modes 5 and 6
PCACoordsP1 = U[:,4:6].T @ P1data
PCACoordsP2 = U[:,4:6].T @ P2data

plt.figure()

plt.plot(PCACoordsP1[0,:],PCACoordsP1[1,:],'d',color='k',label='Person 2')
plt.plot(PCACoordsP2[0,:],PCACoordsP2[1,:],'^',color='r',label='Person 7')

plt.legend()
plt.show()


# %%
## Step 7:  Show eigenface reconstruction of image that was omitted from test set


plt.figure()
testFace = faces[:,np.sum(nfaces[:36])] # First face of person 37
plt.imshow(np.reshape(testFace,(m,n)).T)
plt.set_cmap('gray')
plt.title('Original Image')
plt.axis('off')
plt.show()


#%% Step 8: Reconstruct a dog from photos of people!
#
# Vector "Dog" needs to be reshaped for some reason...
Dog=np.reshape(Dog,(32256,))
testFace = Dog - avgFace
k_list = [50, 200, 500, 1000, 1600]

plt.figure()

fig,axs = plt.subplots(2,3)
axs=axs.ravel()

axs[0].imshow(np.reshape(Dog,(m,n)).T)
axs[0].axis('off')
for i in range(5):
    reconFace = avgFace + U[:,:k_list[i]]  @ ( U[:,:k_list[i]].T @ testFace)
    axs[i+1].imshow(np.reshape(reconFace,(m,n)).T)
    axs[i+1].axis('off')
```

Project: Eigenfaces

Description of the data: `Math350Homework.mat`

- Matrix X that is 24138 26 (26 photos, each with 162 rows and 149 columns)
- Constants $m = 162$ and $n = 149$.

If you want to visualize the photos, you would reshape the corresponding column vector. For example, to visualize the first four faces and put them together in one figure, we would type:

```
for jj=1:4
   subplot(2,2,jj)
   imagesc(reshape(X(:,jj),m,n));
   axis off; axis equal; colormap(gray)
   end
```

Our project is to use the provided code and previous example to do some analysis of this data set.

1. Compute the mean vector and represent it as a face (by plotting it).
2. Compute the first four eigenfaces (and represent them as faces). Put them together in one plot. In a second plot, show what the "photographic negatives" of the eigenfaces look like.
3. What might be a good approximation to the rank of the matrix X (if our goal is to make the faces recognizable, about 74% of the variance).
4. Plot the reconstruction of a randomly selected face using rank 2, 5, 10 and 15. Plot these (as photos) together in one figure.
5. Plot the data using the best two dimensional representation using a new figure. Plot the first 13 data points as asterisks and the remaining 13 as triangles.

Chapter 6

A Best Nonorthogonal Basis

In this section, we examine a particular question whose solution will involve getting an optimal *nonorthogonal* basis, which is quite contrary to our earlier chapters- in fact, the reader should ask why we would ever want to use a non-orthogonal basis when it would be quite easy (using Gram-Schmidt) to construct an orthogonal version of the same basis.

To answer this question, consider the synthetic data example in Figure 6.1. Here there is a definite "natural" basis appearing in the data- and the basis vectors are not orthogonal. While the data was synthetic, we do get similar types of data appearing in the problem of *Blind Signal Separation*. Consider the following tasks:

1. We have a patient that is pregnant. Our overall goal is to listen to the fetus heartbeat, but when we try, the sound of the mother's heartbeat is mixed with the heartbeat of the fetus. Symmetrically, if we were to try to listen to the heartbeat of the mother, we would also hear the heartbeat of the fetus. Is it possible to gather these sounds on microphones and manipulate the data so that the mother's (or fetus) heartbeat has been isolated?

2. We have two microphones placed at random, but distinct, places in a room. We also have two people speaking in the room (the placement of the people is distinct from the placement of the microphones- we do not assume that each microphone is placed in front of each speaker). Is it possible to manipulate the two mixtures of voices so that we can isolate each speaker's voice?

The answer lies in a fairly new technique called *Independent Component Analysis* (ICA) (versus what we studied earlier, *principal components analysis*, or PCA). In ICA, we assume that we have some underlying, statistically independent, processes and that we are observing mixtures of these processes. Our goal is to separate the mixtures.

This problem is also known as *Blind Signal Separation*, where we assume some unknown mixture of signals, and we attempt at separating them.

This process (the problem and the solution) can also be framed in other terms- we will focus on the geometric meaning of the problem, and will solve it using the techniques of linear algebra.

6.1 Set up the Signal Separation Problem

We will assume that there exists a "clean" separation of our observed mixture of two signals (we will be explicit in what we mean by that momentarily, and we will discuss the more general case in a moment). These signals, as time series, are two columns of a matrix S, so that $S \in \mathbb{R}^{p\times 2}$, where p is the length of the sample.

We will further assume that the mixtures we are observing are *linear* mixtures, so that the mixtures we observe may be modeled as:

$$\boldsymbol{x}_1 = \alpha_1 \boldsymbol{s}_1 + \alpha_2 \boldsymbol{s}_2$$

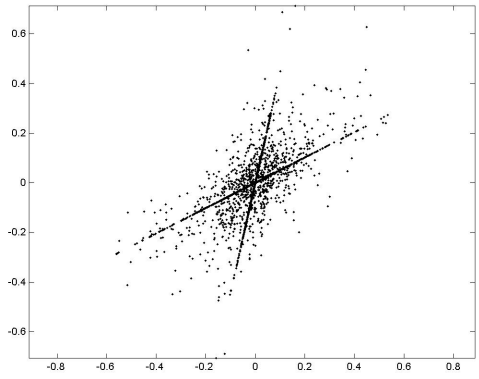

Figure 6.1: A synthetic data example of a naturally emerging set of basis vectors from data- These are not orthogonal.

$$\boldsymbol{x}_2 = \alpha_3 \boldsymbol{s}_1 + \alpha_4 \boldsymbol{s}_2$$

so that $\boldsymbol{x}_1$ is the observed mixture in microphone 1, and $\boldsymbol{x}_2$ is the observed mixture in microphone 2. In linear algebra terms, we can state the problem as follows:

Given $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ as columns of $X \in \mathbb{R}^{p\times 2}$, solve the following equation for $A \in \mathbb{R}^{2\times 2}$ and $S \in \mathbb{R}^{p\times 2}$:

$$X = SA$$

We assume that the rows of X have been mean-subtracted.

If you look at this equation, something should be occurring to you- this is not a well defined problem! There are an infinite number of solutions for A, S. In fact, one solution would be to let A be the 2×2 identity matrix, and $S = X$.

We could also give a solution in terms of the SVD of X, which is what we would do in Principal Component Analysis:

$$X = U_x \Sigma_x V_x^T$$

so that $S = U_x$, and $A = \Sigma_x V_x^T$. In this case the data is "separated" in the sense that the columns of U_x are orthogonal (or *uncorrelated*). Figure 6.2 shows the result of this operation on our synthetic data. It also shows that the desired basis vectors are still rotated.

This process, while not the desired one, is a good first step, but somehow we need the signals to be *independent*, and not just uncorrelated.

Alternatively, let us consider the SVD of the unknown mixing matrix A, $A = U_m \Sigma_m V_m^T$. The problem will be solved if we knew this matrix, as S could be computed by using the inverse of A (we assume that A is full rank- see the exercises for a discussion). Let's try some sample computations now by taking our original equation and substituting the SVD of A:

$$X = SA \Rightarrow X = SU_m \Sigma_m V_m^T$$

Now compute the 2×2 covariance of X:

$$X^T X = V_m \Sigma_m U_m^T S^T S U_m \Sigma_m V_m^T$$

Now, if the rows of S are statistically independent, then they certainly should be uncorrelated. We will assume therefore that SS^T is some scalar multiple of the identity:

$$SS^T = cI_{2\times 2}$$

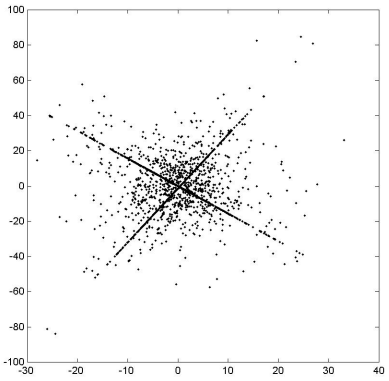

Figure 6.2: The synthetic data set after the SVD transformation as described in the text. In this case, the principal components analysis has left the desired basis vectors in a rotated position. We require one more rotation (multiplication by an orthogonal matrix) to get the desirable results.

which also assumes that the variances of the signals are the same. In particular, we'll assume that the signals in S have been scaled so that $S^T S = I$. In the exercises, we will examine this assumption in more detail.

Using this, we see that:

$$X^T X = V_m \Sigma_m^2 V_m^T = V_x \Sigma_x V_x^T$$

This tells us that V_m and Σ_m are recoverable from the SVD of X: If

$$X = U_x \Sigma_x V_x^T$$

then

$$\Sigma_m = \Sigma_x^{1/2}, \qquad V_m = V_x$$

If we were to stop here and take:

$$Y = X V_x \Sigma_x^{-1/2} = S U_m \Sigma_m V_m^T V_x \Sigma_x^{-1/2} = S U_m$$

we obtain the standard PCA solution. However, the signals are still rotated. We cannot perform another covariance computation on Y, since now:

$$Y^T Y = U_m^T S^T S U_m = I_{2\times 2}$$

How can we compute U_m? There is some justification for what we're about to do- Let's do it first and then discuss it.

Define dA to be the difference matrix for the matrix A. That is, if A has been organized so that it comprises p samples of k time series of data, then let A be $p \times k$. We compute the difference as:

$$dA = A(2:p,:) - A(1:p-1,:)$$

so that dA is now $p-1 \times k$ and

$$(dA)_{ij} = A_{i+1,j} - A_{i,j}$$

If the data in A were the sample of some differentiable function, then dA is an approximation to the derivative using $\Delta t = 1$.

Note that the following matrix equation holds:

$$X = SA \quad \Rightarrow \quad dX = dS\, A$$

so that

$$dX^T dX = A^T\, dS^T dS\, A$$

More particularly, let

$$Y = XV_x\Sigma_x^{-1/2},\ \text{or}\ \ dY = dXV_x\Sigma_x^{-1/2} = dS\, U_m$$

so that $dY^T dY = U_m^T dS^T dS\, U_m$. We now make our second assumption: While $S^T S = I$, we assume that $dS^T dS \neq I$, but is diagonal. In this case, we can recover U_m from the SVD of dY:

$$dY = U_y\Sigma_y V_y^T$$

and $U_m = V_y^T$. Note that the singular vectors are transposed when making this computation!

This finishes our problem, since we have now computed the SVD of the mixing matrix A. The clean signal is now found by taking:

$$S = YV_y$$

We are approximating the mixing matrix by:

$$A \approx V_y^T\Sigma_x^{1/2}V_x = U_m\Sigma_m V_m^T$$

so that the approximate inverse is found via:

$$A^{-1} = V_m\Sigma_m^{-1}U_m^T = V_x^T\Sigma_x^{-1/2}V_y$$

This process of two SVDs, one on the matrix of data X, and another on the data dY can be brought together as a single command. In fact, this process is equivalent to using the *Generalized Singular Value Decomposition*, which will sometimes go under the name of *Quotient Singular Value Decomposition* (GSVD or QSVD, respectively). This simplifies the coding so that you only have to use the following Matlab commands. Let X be the $p \times k$ matrix of k mixtures of signals (this is transposed from our earlier notation). Then signal separation is simply:

```
dX=diff(X);
[U,V,B,C,S]=gsvd(X,dX,0);
```

where the clean mixtures are in the columns of U.

We'll try out both versions in the examples below, then in the next section we'll define the GSVD.

Example

In this example, we will take three columns of data. The first two columns will comprise a circle and the last will be white noise (that is, random data from a normal distribution). We will then multiply this by 3×3 mixing matrix A, which was originally taken so that the elements are from a normal random variable then subsequently hard coded for you to replicate. Denote the mixed data matrix as X, as we did previously. Note that with column-wise data, the mixing matrix equations become:

$$X = SA \Rightarrow X = SU\Sigma V^T$$

where we try to determine U, Σ, V.

Here is the script file to produce the signal separation:

```
%Script file to produce Example 1, ICA
numpts=400;
t=linspace(0,3*pi,numpts);
S=[cos(t'), sin(t'), randn(numpts,1)]; %Separated Signals
A =[  -0.0964   -0.1680    1.6777
    0.4458    0.1795    1.9969
   -0.2958    0.4211    0.6970]; %Mixing Matrix
```

```
X=S*A;

dX=diff(X);
[U,V,B,C,S] = gsvd(X,dX,0); %Clean Signal in U

%The double SVD code, equivalent to the GSVD:
[Ux,Sx,Vx]=svd(X,0);
Y=X*Vx*diag(1./sqrt(diag(Sx)));  %Stopping here is basic PCA
dY=diff(Y);
[Uy,Sy,Vy]=svd(dY,0);
S2=Y*Vy;  %S2 is also the clean signal


%Plotting routines below:
figure(1)
for j=1:3
   subplot(3,1,j)
   plot(U(:,j));  %Clean Signals from GSVD
end
figure(2)
for j=1:3
   subplot(3,1,j);
   plot(S2(:,j));  %Clean Signals from double SVD
end
figure(3)
for j=1:3
   subplot(3,1,j)
   plot(Y(:,j));  %Results of PCA (or KL)
end
```

6.2 Signal Separation of Voice Data

In this example, we will linearly mix two voice signals, then separate them using the techniques we previously described. This example is best done using a computer with a good sound card, but can be done without it.

Matlab comes with several sound files. For this example, we will use `handel` (a sample of the chorus to Handel's "Messiah"), and `laughter` (a sample of people laughing). The two files have different lengths, so we'll have to cut the longer file off so that they match.

Here is the Matlab code:

```
%Script for sound files
load handel
y1=y;
load laughter
S=[y y1(1:52634)];  %Clean samples in the columns of S
A =[ -0.5883   -0.1364; 2.1832    0.1139]; %Mixing Matrix
X=S*A;

mX=mean(X);
X=X-repmat(mX,52634,1);

figure(1)
plot(X(:,1),X(:,2),'.');
```

```
title('Mixed Signals');

%For comparison purposes, here's the SVD
[Ux,Sx,Vx]=svd(X,0);
Y=X*Vx*(1./sqrt(diag(Sx)));

figure(2)
plot(Y(:,1),Y(:,2),'.');
title('KL Results')
%Listen to the results:  You'll still hear mixtures
%soundsc(Y(:,1));
%soundsc(Y(:,2));

dX=diff(X);
[Y2,V,B,C,S3]=gsvd(X,dX,0);
figure(3)
plot(Y2(:,1),Y2(:,2),'.');
title('ICA Results');
%Listen to the results:  They will be clean!
%soundsc(Y2(:,1));
%soundsc(Y2(:,2));
```

More to Think About: There are a lot of experiments you can try with this data. Here are some things you might try:

- Plot the signals as time series if you've never looked at voice data before. Also plot the differenced signal. You might also look at the histograms of the two voice signals using `hist`. Do the signals look like they are normally distributed or do they follow a Laplace distribution?
- Change the mixing matrix to a random matrix. Will you always get good results?
- Listen to the difference matrix dX. Does it still sound like the original? Listen to the second, third, fourth difference. Why does the "derivative" of the signal sound just like the original (perhaps with a different timbre quality, but recognizable just the same)?
- Check the assumptions on the clean signal S and the differenced signal dS- Are the assumptions met?

6.3 A Closer Look at the GSVD

Suppose that we have two matrices $X \in \mathbb{R}^{m\times n}$ and $Z \in \mathbb{R}^{p\times n}$. The GSVD of matrices X, Z is a decomposition where we determine $\hat{U}, \hat{V}, W, C, S$ so that:

$$X = \hat{U}CW^T \qquad Z = \hat{V}SW^T$$

where $\hat{U}, \hat{V}$ have orthonormal columns, C, S are diagonal matrices such that $C^TC + S^TS = I$, and W is an invertible matrix. In Matlab, the command is:

```
[Ux,Vz,W,C,S]=gsvd(X,Z)
```

The values of C, S and W satisfy the following generalized eigenvalue problem:

$$s_i^2 A^T A w_i = c_i^2 B^T B w_i$$

The solution we use for the signal separation is now either:

$$X = \hat{U}CW^T = \hat{U}(CW^T) = SA \text{ or } X = \hat{U}CW^T = (\hat{U}C)W^T = SA$$

In the special case that $s_i \neq 0$, we'll show that we can find x_i using two regular SVD's as we did in the signal separation.

In this case, the eigenvector problem can be written as:

$$X^T X w_i = \lambda Z^T Z w_i \Rightarrow \left(X^T X - \lambda Z^T Z\right) w_i = 0$$

If we let $X = U\Sigma_x V_x^T$ be the SVD of X, then we can rewrite this as:

$$\left(V_x \Sigma_x^2 V_x^T - \lambda Z^T Z\right) x_i = 0$$

This can be factored as:

$$\left[V_x \Sigma_x \left(I - \lambda(\Sigma_x^{-1} V_x^T) Z^T Z (V_x \Sigma_x^{-1})\right) \Sigma_x V_x^T\right] w_i = 0$$

or equivalently, if V_x is square and Σ_x is invertible:

$$\left(I - \lambda(\Sigma_x^{-1} V_x^T) Z^T Z (V_x \Sigma_x^{-1})\right) \Sigma_x V_x^T w_i = 0$$

If we let $Y = Z V_x \Sigma_x^{-1}$, and $q_i = \Sigma_x V_x^T w_i$, the previous equation can be written as:

$$(I - \lambda Y^T Y) q_i = 0$$

Therefore, q_i is an eigenvector of $Y^T Y$, or a right singular vector of Y. We can compute $w_i = V_x \Sigma_x^{-1} q_i$, or

$$W = V_x \Sigma_x^{-1} Q = V_x \Sigma_x^{-1} V_y$$

To summarize, the GSVD is equivalent to two SVDs as follows:

- Let $X = U\Sigma_x V_x^T$ be the SVD of A
- Let $Y = Z V_x \Sigma_x^{-1}$ be a "whitening" transformation.
- Let $Y = U_y \Sigma_y V_y^T$ be the second SVD.
- Final answer: $W = V_x \Sigma_x^{-1} V_y$

To connect this process to our previous signal separation solution, let's recall what we did there: Let X be the mixed signal, dX be the differenced signal (we are now thinking of $Z = dX$):

- Let $X = U_x \Sigma_x V_x^T$ be the SVD of X.
- Let $dY = dX V_x \Sigma_x^{-1/2}$ be the "whitened" signal.
- Let $dY = U_y \Sigma_y V_y^T$ be the second SVD.
- Then $S = Y V_y = X V_x \Sigma_x^{-1/2} V_y = XW$ is the clean signal.