Homework 1

Nate Mankovich
Math Modeling

February 8, 2018

1 Written Portion

Problem 1. Consider the vector whose components are given w.r.t. the standard basis are

$$x = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}$$

Determine the coordinates of this point w.r.t. the Walsh basis

$$U_1 = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

and the Haar basis

$$U_2 = \frac{1}{2}\begin{bmatrix} 1 & 1 & \sqrt{2} & 0 \\ 1 & 1 & -\sqrt{2} & 0 \\ 1 & -1 & 0 & \sqrt{2} \\ 1 & -1 & 0 & -\sqrt{2} \end{bmatrix}$$

Which representation do you think is better for this point x and why? You may use MATLAB or another computer program to assist in your calculations.
I used MATLAB to calculate the inverses of U_1 and U_2. Then I computed

$$U_1^{-1}x = \begin{bmatrix} 0 \\ 0 \\ -2 \\ 2 \end{bmatrix} \text{ and } U_2^{-1}x = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1.412 \end{bmatrix}.$$

In my opinion, the Harr basis is a better representation of the data point x because the magnitude of x in terms of U_2 is smaller than the magnitude of x in terms of U_1.

Problem 2. Consider the two-dimensional subspace spanned by the vectors

$$U = \frac{1}{4}\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \\ 1 & -1 \end{bmatrix} = [u_1|u_2]$$

Compute the projection of the point

$$x = \begin{bmatrix} 2 \\ 3 \\ 1 \\ -1 \end{bmatrix}$$

onto the subspace spanned by U. What is the novelty of this point? What is the representation of x in the 2D subspace w.r.t. the basis for the subspace?

$$\bar{x} = \sum_{i=1}^{2} \frac{\langle x, u_i \rangle}{||u_i||} u_i = \begin{bmatrix} 2.5 \\ 2.5 \\ 0 \\ 0 \end{bmatrix}$$

The MATLAB code this projection is:

```
for i=1:2
xbar = xbar + ((transpose(x)*U(:,i))/(norm(U(:,i))^2))*U(:,i);
end
```

The representation of the projection of x with respect to the basis defined by the columns of U is

$$\bar{x} = 5u_1 + 5u_2$$

Problem 3. Use the QR algorithm to compute an orthonormal basis from the vectors

$$x_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, x_2 = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}, x_3 = \begin{bmatrix} 0 \\ -1 \\ -1 \end{bmatrix}$$

Verify that $Q^T Q = I$. Work through this problem by hand and verify via MATLAB.

$$q_1 = \frac{x_1}{||x_1||} = \sqrt{14} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.27 \\ 0.53 \\ 0.80 \end{bmatrix}$$

$$q_2 = \frac{(I - q_1 q_1^T)x_2}{||(I - q_1 q_1^T)x_2||} = \begin{bmatrix} 0.9 & -0.1 & -0.2 \\ -0.1 & 0.7 & -0.4 \\ -0.2 & -0.4 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix} \Bigg/ \Bigg|\Bigg| \begin{bmatrix} 0.9 & -0.1 & -0.2 \\ -0.1 & 0.7 & -0.4 \\ -0.2 & -0.4 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix} \Bigg|\Bigg|$$

$$= \begin{bmatrix} 0.5 \\ -2.8 \\ 1.7 \end{bmatrix} \Bigg/ \Bigg|\Bigg| \begin{bmatrix} 0.5 \\ -2.8 \\ 1.7 \end{bmatrix} \Bigg|\Bigg| = \begin{bmatrix} 0.2 \\ -0.8 \\ 0.5 \end{bmatrix}$$

$$q_3 = \frac{(I - q_1 q_1^T)(I - q_2 q_2^T)x_3}{||(I - q_1 q_1^T)(I - q_2 q_2^T)x_3||}$$

$$= \begin{bmatrix} 1.0 & 0.1 & -0.1 \\ 0.1 & 0.3 & 0.4 \\ -0.1 & 0.4 & 0.7 \end{bmatrix} \begin{bmatrix} 0.9 & -0.1 & -0.2 \\ -0.1 & 0.7 & -0.4 \\ -0.2 & -0.4 & 0.3 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ -1 \end{bmatrix} \Bigg/ \Bigg|\Bigg| \begin{bmatrix} 1.0 & 0.1 & -0.1 \\ 0.1 & 0.3 & 0.4 \\ -0.1 & 0.4 & 0.7 \end{bmatrix} \begin{bmatrix} 0.9 & -0.1 & -0.2 \\ -0.1 & 0.7 & -0.4 \\ -0.2 & -0.4 & 0.3 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ -1 \end{bmatrix} \Bigg|\Bigg|$$

$$= \begin{bmatrix} 0.3 \\ 0 \\ -1 \end{bmatrix} \Bigg/ \Bigg|\Bigg| \begin{bmatrix} 0.3 \\ 0 \\ -1 \end{bmatrix} \Bigg|\Bigg| = \begin{bmatrix} 0.9 \\ 0 \\ -0.3 \end{bmatrix}$$

I verified in MATLAB that $QQ^T = I$.

Problem 4. Recall that the eigenvectors of symmetric matrices associated with distinct eigenvectors are orthogonal and that eigenvalues are real. Prove these facts. (Try without looking at your old linear algebra book.)
Suppose M is a symmetric matrix. This means $M = M^T$. Suppose there are r eigenvectors of M $\{v_i\}_{i=1}^r$ associated with r distinct nonzero eigenvalues $\{\lambda_i\}_{i=1}^r$. We will use the inner product to prove the eigenvectors are orthogonal and that eigenvalues are real.

$$\lambda_i\langle v_i, v_i\rangle = \langle \lambda_i v_i, v_i\rangle = \langle Mv_i, v_i\rangle = \langle v_i, M^T v_i\rangle = \langle v_i, Mv_i\rangle = \langle v_i, \lambda_i v_i\rangle = \bar{\lambda_i}\langle v_i, v_i\rangle$$

So $\bar{\lambda_i} = \lambda_i$ which means $\text{im}(\lambda_i) = 0$ so $\lambda_i \in \mathbb{R}$ for $0 < i \leq r$.
Now we will show the eigenvectors are orthogonal, aka $\langle v_i, v_j\rangle = 0 \forall i \neq j$. Suppose $0 < i, j \leq r$ and $i \neq j$.

$$\langle Mv_i, v_j\rangle = \langle v_i, Mv_j\rangle \implies \langle \lambda_i v_i, v_j\rangle = \langle v_i, \lambda_j v_j\rangle$$

$$\implies \lambda_i\langle v_i, v_j\rangle = \bar{\lambda_j}\langle v_i, v_j\rangle = \lambda_j\langle v_i, v_j\rangle \text{ because } \lambda \in \mathbb{R}$$

Since the eigenvalues are distinct $\langle v_i, v_j\rangle = 0 \forall i \neq j$ so the eigenvectors are distinct.

Problem 5. In class we showed that the optimization problem for the first principal vector is given by

$$\max \phi^T C\phi - \lambda(\phi^T\phi - 1) \text{ (where } \dim(C) = n)$$

leads to the necessary condition

$$C\phi = \lambda\phi$$

for the best eigenvector. Provide the details of this calculation.
We can maximize $\phi^T C\phi - \lambda(\phi^T\phi - 1)$ by setting the partial derivatives with respect to ϕ_k equal to zero for all $k = 1, 2, .., n$.

$$\frac{\partial}{\partial\phi_k}\big(\sum_{j=1}^{n}\sum_{i=1}^{n}\phi_i\phi_j c_{ij} - \lambda\big(\sum_{i=1}^{n}\phi_i^2 - 1\big)\big) = 0 \ \forall\, k \in \mathbb{N},\ k \leq n$$

$$\implies \phi_1 c_{k1} + \phi_2 c_{k2} + ... + 2\phi_k c_{kk} + ... + \phi_n c_{kn} + \phi_1 c_{1k} + \phi_2 c_2 k + ... + \phi_n c_{nk} = 0 \ \forall\, k \in \mathbb{N},\ k \leq n$$

Since C is hermetian $c_{ij} = c_{ji}$ for all $i, j \in \mathbb{N}$ and $i, j \leq n$ which means this nasty sum us really just:

$$2\sum_{i=1}^{n}\phi_i c_k i = 2\lambda\phi_k \implies \sum_{i=1}^{n}\phi_i c_k i = \lambda\phi_k \implies C\phi = \lambda\phi$$

Problem 6. Let $\{x^\mu\}$ be ant data set in $\mathbb{R}^N$ and let $\beta = \{\}$ be a bais. Further, for any pattern $x^{(\mu)}$ write

$$x^{(\mu)} = \sum_{i=1}^{N}\alpha_i^{(\mu)}u^i$$

(a) Show that if the data set has been mean subtracted then, for each i,

$$\sum_{\mu=1}^{N}\alpha_i^{(\mu)} = 0$$

(b) Show that the statistical variance of $\alpha_i^{(\mu)}$ over μ for fixed i is the eigenvalue λ_i, of the covariance matrix C.
(a) Since the data is mean subtracted we have

$$\frac{1}{p}\sum_{\mu=1}^{p}x^{(\mu)} = 0$$

$$\implies \frac{1}{p}\sum_{\mu=1}^{p}\Big(\sum_{i=1}^{N}\alpha_i^{(\mu)}u^i\Big) = 0$$

$$\implies \sum_{\mu=1}^{p}\sum_{i=1}^{N}\alpha_i^{(\mu)}u^{(\mu)} = 0 \implies \sum_{\mu=1}^{p}\Big(\sum_{i=1}^{N}\alpha_i^{(\mu)}\Big)u^{(\mu)} = 0$$

This is a linear combination of basis elements of β equal to zero. Since a basis is linearly independent this means

$$\sum_{i=1}^{N}\alpha_i^{(\mu)} = 0\ \forall\mu$$

(b) Again we will assume the data has been mean subtracted.
Notice if the mean of the data ensemble is 0 than so is the mean of $\alpha_i^{(\mu)}$.

$$\text{mean}(\alpha_i^{(\mu)}) = \frac{1}{p}\sum_{\mu=1}^{p}\alpha_i^{(\mu)} = \frac{1}{p}\sum_{\mu=1}^{p}\langle u^{(i)}, x^{(\mu)}\rangle$$

$$= \frac{1}{p}\sum_{\mu=1}^{p}{u^{(i)}}^T x^{(\mu)} = \frac{{u^{(i)}}^T}{p}\sum_{\mu=1}^{p}x^{(\mu)} = {u^{(i)}}^T\text{mean}(x^{(\mu)}) = 0$$

Then the definition of statistical variance over μ of $\alpha_i^{(\mu)}$ for a fixed i is

$$\frac{1}{p}\sum_{\mu=1}^{p}(\alpha_i^{(\mu)} - \text{mean}(\alpha_i^{(\mu)}))^2 = \frac{1}{p}\sum_{\mu=1}^{p}{\alpha_i^{(\mu)}}^2$$

$$= \frac{1}{p}\sum_{\mu=1}^{p}\langle u^{(i)}, x^{(\mu)}\rangle^2 = \frac{1}{p}\sum_{\mu=1}^{p}({u^{(i)}}^T x^{(\mu)})^2$$

$$= \frac{1}{p}\sum_{\mu=1}^{p}({u^{(i)}}^T x^{(\mu)}) = {u^{(i)}}^* C u^{(i)} = \lambda_i$$

2 Computation

2.1 Part 1 (pumpkins)

a) Compute the mean pumpkin and subtract it from the data. Does this average look like you would expect? This looks like the average of the other pumpkins. This is what we excpect. It is important to note that we

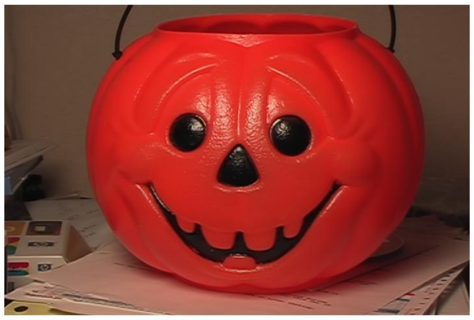

Figure 1: The mean pumpkin.

can use a linear transformation to perhaps improve the visualization of these data. Sadly, this Below is the MATLAB code for this operation:

```
%compute the mean
dataMean = mean(X,2);
%subtract the mean
for i=1:200
    msX(:,i) = double(X(:,i))-dataMean(:);
end
%show the image
imshow(uint8(resize(dataMean,[480,720,3])))
```

b) Compute the best basis using PCA on the mean subtracted data. Include pictures of eigenpumpkins 1-4 and 101-104. (Note that these eigenvectors need to be reassembled into RGB 3-way arrays with values scaled to be in the same form as the initial data, i.e., integers 0, . . . , 255.) Plot the eigenvalues. Comment. Below are the pictures of the eigenpumpkins 1-4 and 101-104:

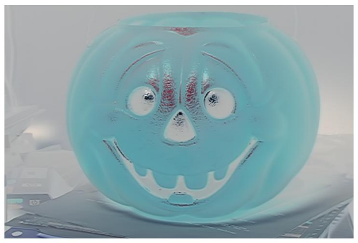
Figure 2: The first eigenpumpkin.

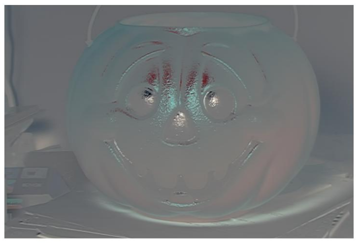
Figure 3: The second eigenpumpkin.

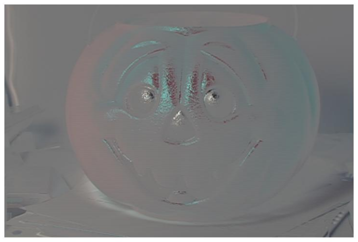
Figure 4: The third eigenpumpkin.

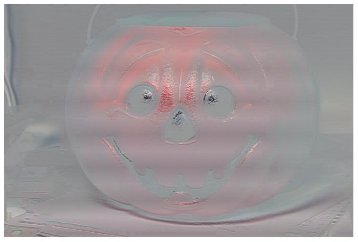
Figure 5: The fourth eigenpumpkin.

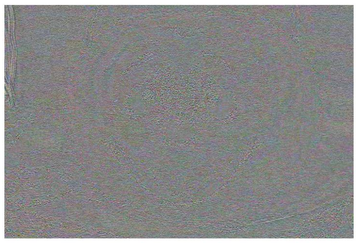
Figure 6: The 101st eigenpumpkin.

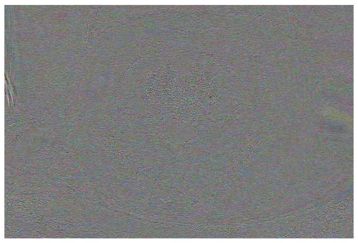
Figure 7: The 102nd eigenpumpkin.

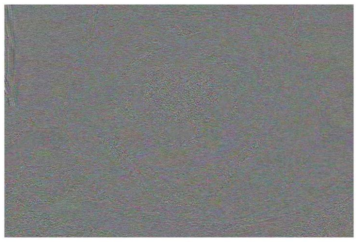
Figure 8: The 103rd eigenpumpkin.

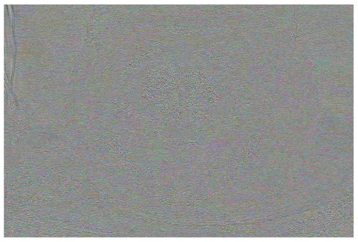
Figure 9: The 104th eigenpumpkin.

Here is a plot of the eigenvalues from the PCA:

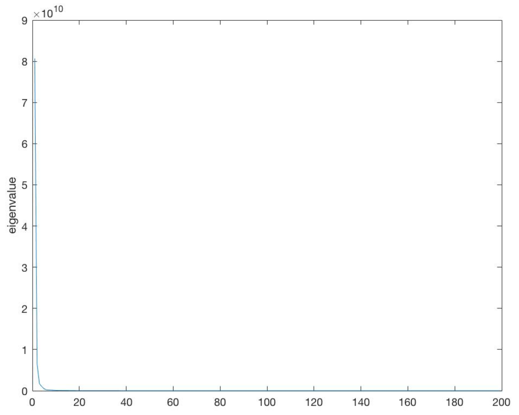

Figure 10: The plot of the pumpkin eigenvalues.

It is surprising how large the eigenvalues are. The eigenvalues go to zero quickly, which implies that we have a low dimensional data set. Below is the code used to produce the eigenvectors, eigenvalues and the eigenvalue plot:

```
[msXevecs,evals] = eig(transpose(msX)*msX);
evecs = msX*msXevecs;

%plot evals
for k=1:200;
    plotevals(201-k) = evals(k,k);
end
plot(1:200,plotevals)
```

c) Repeat part b) using the thin svd, i.e., svd(X,0) and compare the results left singular vectors and squared singular values with the eigenvectors and eigenvalues from PCA.
We do the SVD and found that the eigenvalues are the singular values squared. This is what we expect via the definitions of PCA and SVD.

```
[tU,tS,tV] = svd(msX,0);
```

Then we can plot eigenpumpkins 1-4 and 101-104:

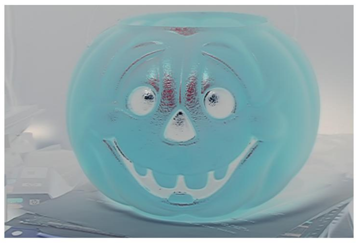

Figure 11: The first eigenpumpkin.

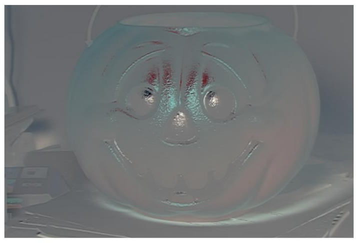

Figure 12: The second eigenpumpkin.

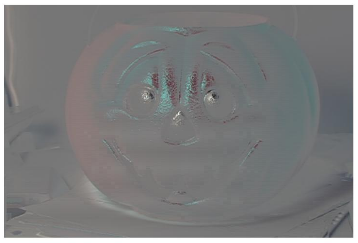

Figure 13: The third eigenpumpkin.

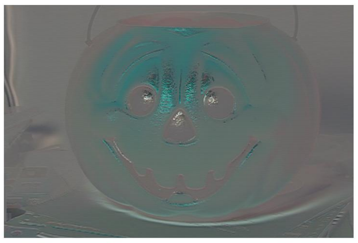

Figure 14: The fourth eigenpumpkin.

We expect the eigenpumkins via the SVD and PCA to be the same. The images are pretty much the same (modulo any minor issues with the rescaling alogorithm).

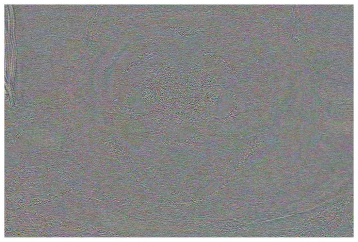

Figure 15: The 101st eigenpumpkin.

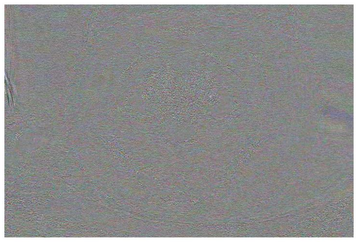

Figure 16: The 102nd eigenpumpkin.

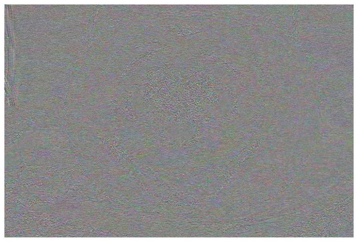

Figure 17: The 103rd eigenpumpkin.

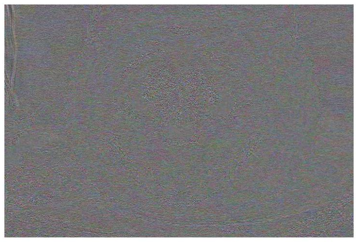

Figure 18: The 104th eigenpumpkin.

As we expected, the squared singular values from the SVD are the eigenvalues using the PCA. Below is a plot of the squared singular values.

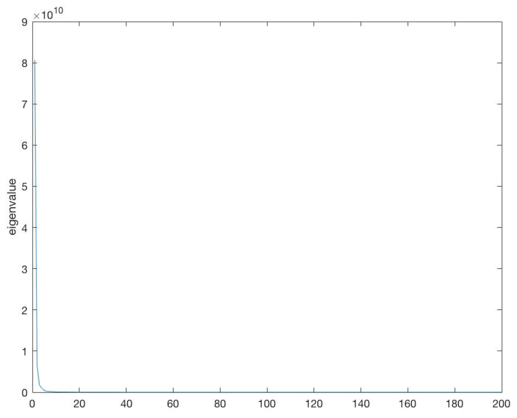

Figure 19: The plot of the pumpkin squares singular values.

We need to apply a linear transformation to the eigenpumpkins visualize them. Below is the code to do so:

```
i = 104; %eigenvector number
dmin = min(tU(:,i));
dmax = max(tU(:,i));
fm = [1 dmax; 1 dmin];
b = inv(fm)*[255;0];
image1 = uint8(b(1) + b(2)*tU(:,i));
image = reshape(image1,[480,720,3]);
imshow(image)
```

d) Plot the coefficients of the pumpkin $(\alpha_1^{(\mu)}, \alpha_2^{(\mu)})$ with respect to the PCA basis.
Below is the plot of $(\alpha_1^{(\mu)}, \alpha_2^{(\mu)})$ for $1 \leq \mu \leq 200$ and $\mu \in \mathbb{N}$. $\alpha_1^{(\mu)}$ is on the horizontal axis and $\alpha_2^{(\mu)}$ is on the vertical axis. Below is the code used to generate the plot:

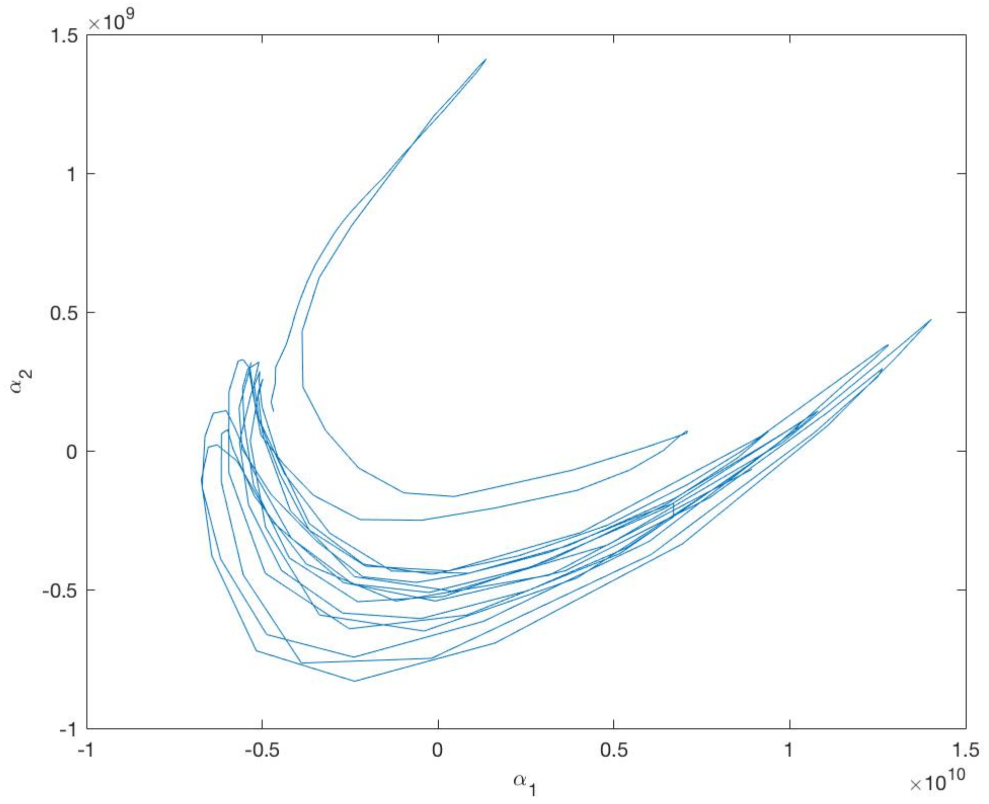

Figure 20: $(\alpha_1^{(\mu)}, \alpha_2^{(\mu)})$

```
%(alpha^mu_1,alpha^mu_2)
%calculate alpha1
for k=1:200
    alpha1(k) = msX(:,k)'*evecs(:,200);
end
%calculate alpha2
for k=1:200
    alpha2(k) = msX(:,k)'*evecs(:,199);
end
%plot alpha1 and alpha2
plot(alpha1(:),alpha2(:))
```

2.2 Part 2 (Fourier data)

This project concerns the application of the KL procedure for incomplete data. Let the complete data set be translationally invariant:

$$f(x_m, t_\mu) = \frac{1}{N}\sum_{k=1}^{N} \frac{1}{k}\sin[k(x_m - t_\mu)],$$

where $m = 1, ..., M$, with M as the dimension of the ambient space and $\mu = 1, ..., P$, with P as the number of points in the ensemble. Let $x_m = (m-1)2\pi/M$ and $t_\mu = (\mu-1)2\pi/P$. Each pattern in the incomplete ensemble may be written

$$x^{(\mu)} = m^{(\mu)} f^{(\mu)}$$

where $(f^{(mu)})_m = f(x_m, t_\mu)$. Let $P = M = 64$ and $N = 3$.

Part A. What is the rank of the data matrix? Provide a theoretical argument followed by an actual computation in Matlab.
First we write MATLAB code to generate the data.

```
%Generate the ensemble. Call the data ensemble f.
M=64;
P=64;
N=3;
for m=1:M
    xm = (m-1)*2*pi/M;
    for mu=1:P
        tmu = (mu-1)*2*pi/P;
        f1=0;
        for k=1:N
            f1 = (1/k)*sin(k*(xm-tmu))+f1;
        end
        f(m,mu) = N*f1;
    end
end
```

The rank of f should be 6 due to the periodicity of sine and cosine. That is to day, it is known that sine and cosine are orthogonal via the standard continuous inner product as an integral from 0 to 2π. Suppose x is a variable and a is a fixed real number. Now we can make use of the identity

$$\sin(x-a) = b\sin(x) + c\cos(x)$$

Since we are defining the coordinates of our data as a sum of three different terms of the form $\sin(x-a)$. We can see the dimension of our data set is 3 times 2 which is 6.
Then we can compute the rank of f in MATLAB using

```
rank(f)
```

This shows that the rank of f is indeed 6.

Part B. Compute a best basis for this data. Does it look familiar?
We use the SVD to calculate the best basis for the data.

```
[fU,fS,fV] = svd(f);
```

A plot of the basis vectors suggests we have a Fourier basis. This makes sense because out data is generated via sine and cosine.

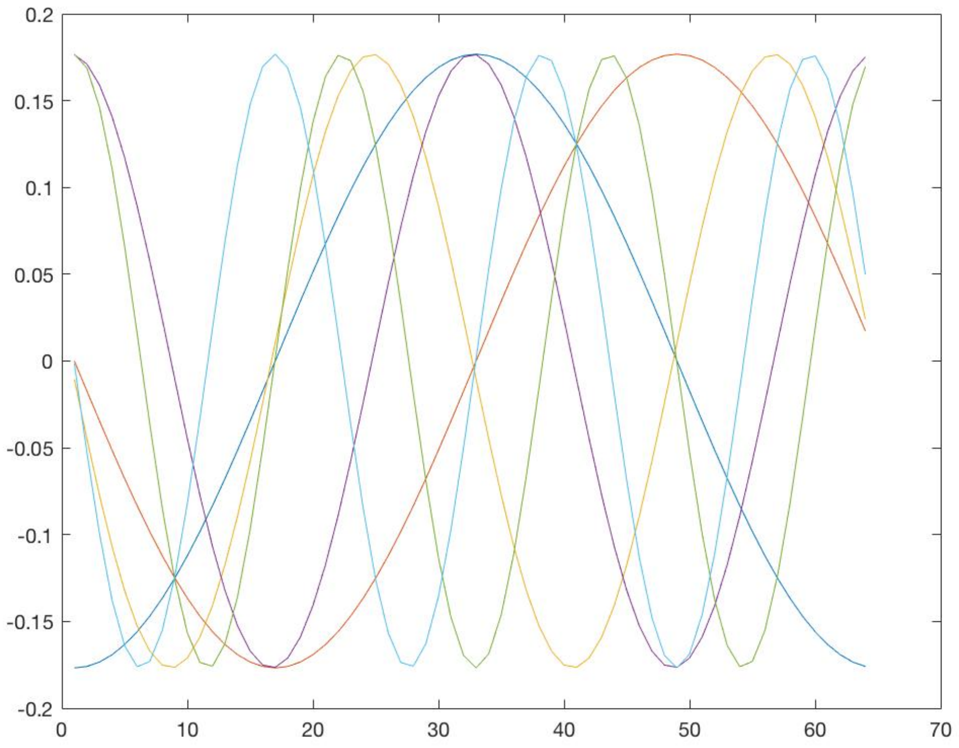

Figure 21: These are the six linearly independent basis vectors (left singular vectors associated with nonzero eigenvalues) from the SVD.

Part C. Write a code to repair a single pattern with 50 percent missing entries using this basis. If you increase the percentage of missing data, how far can you go before you fail to reconstruct the pattern?
We can increase the percentage of missing data to about 92 percent before we fail reconstructing the corrupted pattern. Perhaps this is because the rank of f is 6 and any mask that is greater than 92 percent causes our masked pattern to have less than six nonzero values. This is what happens when we go past the 92 percent threshold:

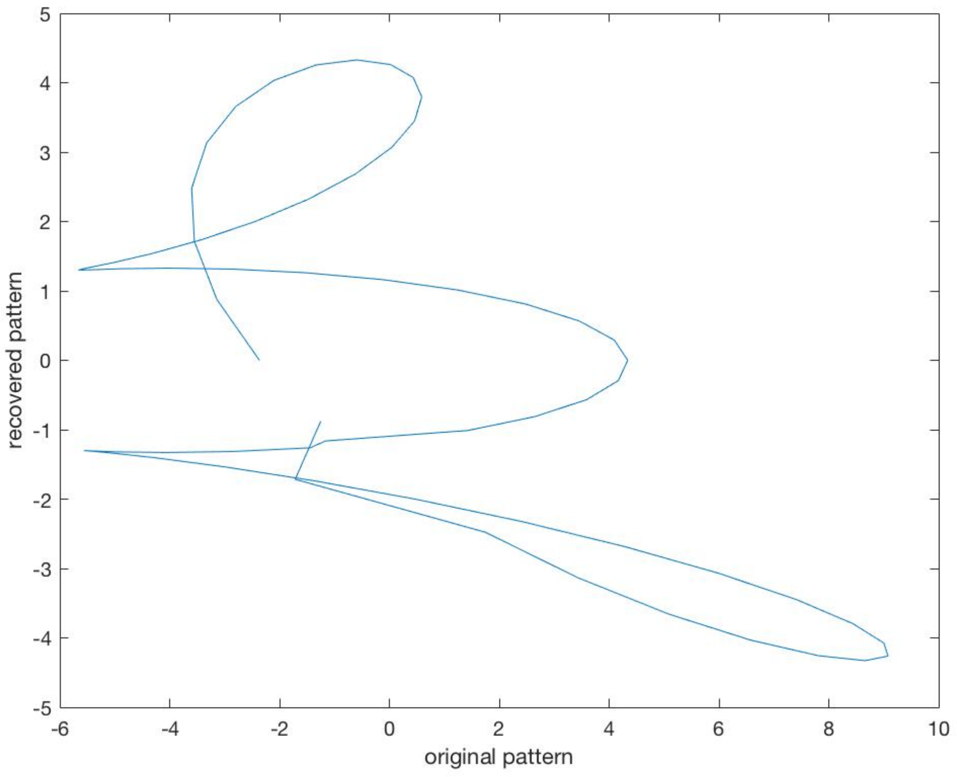

Figure 22: The original pattern versus the repaired pattern at 94 percent mask.

Below is the code for repairing an approximately 94 percent zeroes masked data set assuming the data ensemble is stored in the variable f.

```
%mask test
pct = .94;          %input percentage here

%best basis using svd
[fU,fS,fV] = svd(f);

mask = rand(64,1)>= pct;

%take the first pattern and make it gappy using 50% mask
xg = f(:,1).*mask

%solve Ma = f (a = hat(alpha)).

%compute M in linear system
for i = 1:64
```

```
    for j = 1:64
        M(i,j) = (fU(:,i).*fU(:,j))'*mask;
    end
end

for i = 1:64
    f0(i) = (xg.*fU(:,i))'*mask;
end

f0 = f0';

%NOTE: use D=6 for repair
%6D reconstruction of xg
D=6
M(1:D,1:D);
f0(1:D);
a = inv(M(1:D,1:D))*f0(1:D);
xapp = a(1)*fU(:,1)+a(2)*fU(:,2)+a(3)*fU(:,3)+a(4)*fU(:,4)+a(5)*fU(:,5)+a(6)*fU(:,6);

%create a repair vector
for i=1:64
if mask(i,1) == 0
        fr(i,1) = xapp(i);
    else
        fr(i,1) = f(i,1);
    end
end

%plot
plot(fr(:,1),f(:,1))
%see if equal
isequal(f(:,1),fr(:,1)),xaxis('original pattern'),yaxis('recovered pattern')
```

Part D. Write a code to repair the entire ensemble of corrupted patterns. Assume no good basis is available for the repair.
Supposing f is the data matrix, the code to make the desired repair is as follows:

```
%mask test
pct = .50;          %input percentage here

%generate the mask
mask = rand(64,64)>= pct;

%take the first pattern and make it gappy using 50% mask
fg = f.*mask;

%compute svd
[fU,fS,fV] = svd(fg);

for k = 1:64

    %fill in the approximation with the old svd
    if k < 64
        fapp(:,k+1:64) = fg(:,k+1:64);
    end

    %compute svd
    if i ~= 1
        [fU,fS,fV] = svd(fapp);
    end

    %compute M in linear system
    for i = 1:64
        for j = 1:64
            M(i,j) = (fU(:,i).*fU(:,j))'*mask(:,k);
        end
    end

    for i = 1:64
        f0(i) = (fg(:,k).*fU(:,i))'*mask(:,k);
    end

    f0 = f0';

    %NOTE: use D=6 for repair
    %6D reconstruction of xg
    D=6
    M(1:D,1:D);
    f0(1:D);
    a = inv(M(1:D,1:D))*f0(1:D);
    xapp = a(1)*fU(:,1)+a(2)*fU(:,2)+a(3)*fU(:,3)+a(4)*fU(:,4)+a(5)*fU(:,5)+a(6)*fU(:,6);

    %create a repair vector
    for i=1:64
        if mask(i,k) == 0
            fr(i,k) = xapp(i);
        else
            fr(i,k) = fg(i,k);
        end
    end

    %clear some variables
    clr = {'f0','a','xapp','fU','fS','fV','M'}
    clear(clr{:})

    %begin the reconstruction matrix
    fapp(:,1:k) = fr;
end
```

Part E. Comment on the rate of convergence of the algorithm as you vary the amount of missing data. Use the measure

$$E(k) = \left(\sum_{i=1}^{r} \lambda_i(k) - \lambda_i(k-1)) \right)^{1/2}$$

where r is the rank of the approximation of the gappy data, k is the iteration and the λ are the eigenvalues of the best basis. The algorithm should terminate for the smallest k such that

$$E(k) \leq 0.01$$

The data does not converge after more than 100 iterations for a mask of approximately 70 percent. For percent masks below 70 the graph below displays the number of iterations.

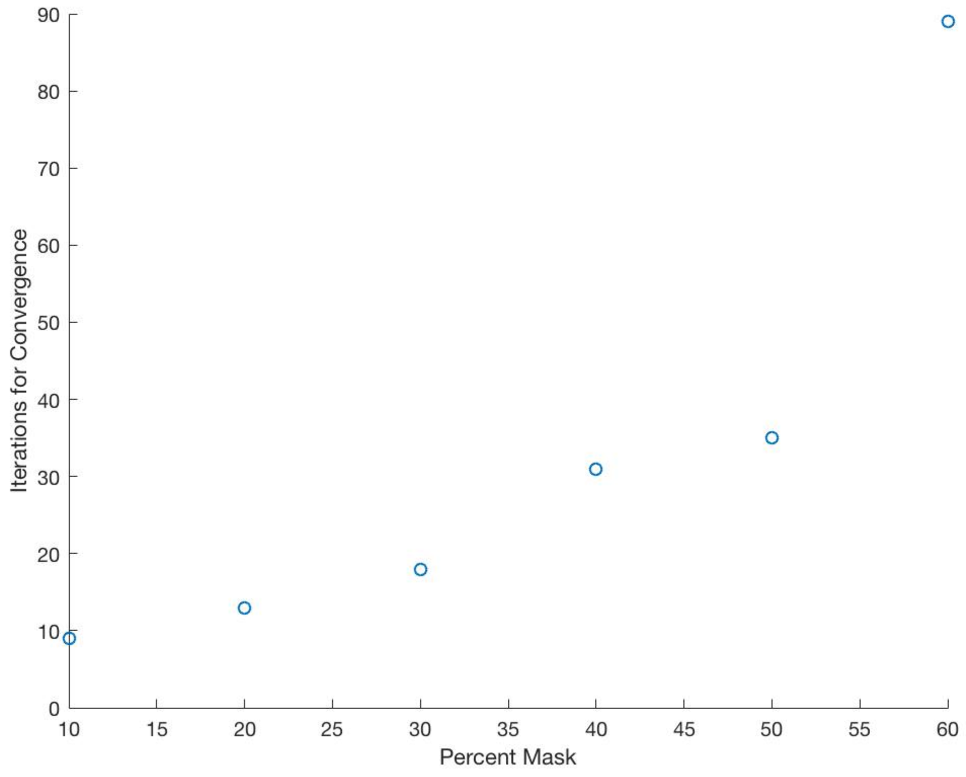

Figure 23: A plot for the number of iterations for convergence using masks at $10, 20, 30, 40, 50$, and 60 percent.

```
%the percent mask
pct = .5

%generate the mask
mask = rand(64,64)>= pct;

%take the first pattern and make it gappy using mask
fg = f.*mask;

%compute svd
[fU,fS,fV] = svd(fg);

fr = fg;

E(1) = 1;
K=2;

while E(K-1) >= 0.01
    for k = 1:64

        %compute svd for first iteration
        if k == 1
            [fU,fS,fV] = svd(fg);
            S(:,1) = diag(fS);
        end

        %compute M in linear system
        for i = 1:64
            for j = 1:64
                M(i,j) = (fU(:,i).*fU(:,j))'*mask(:,k);
            end
        end

        for i = 1:64
            f0(i) = (fr(:,k).*fU(:,i))'*mask(:,k);
        end

        f0 = f0';

        %NOTE: use D=6 for repair 6D reconstruction of xg
        D=6;
        M(1:D,1:D);
        f0(1:D);
        a = inv(M(1:D,1:D))*f0(1:D);
        xapp = a(1)*fU(:,1)+a(2)*fU(:,2)+a(3)*fU(:,3)+a(4)*fU(:,4)+a(5)*fU(:,5)+a(6)*fU(:,6);

        %create a repair vector
        for i=1:64
            if mask(i,k) == 0
                fr(i,k) = xapp(i);
            else
                fr(i,k) = fg(i,k);
            end
        end
        %clear some variables
        clr = {'f0','a','xapp','M'};
        clear(clr{:});

        %begin the reconstruction matrix
        fapp(:,1:k) = fr(:,1:k);

        if k ~= 64
            fapp(:,k+1:64) = fr(:,k+1:64);
        end

        %compute svd
        [fU,fS,fV] = svd(fr);
        S(:,K) = diag(fS);

    end
    %calculate the error
    E(K) = 0;
    for i=1:6
        E(K) = E(K)+(S(i,K)^2-S(i,K-1)^2)^2;
        i
    end
    E(K) = E(K)^(1/2)
    K=K+1
end
```