Math 521 Homework 4

Due on Thursday, April 19, 2018

Kristin Holmbeck

Contents

List of Figures

Theory

1. Eigen-relationship

Consider the two eigenvector problems

$$C_x \boldsymbol{u} = \lambda_x \boldsymbol{u}$$

and

$$C_s \boldsymbol{v} = \lambda_s \boldsymbol{v}$$

where the matrices are related by $C_x = C_s + \alpha I$, where α is a real number and I is the usual identity matrix. Show that if $\boldsymbol{u}$ is an eigenvector of C_x, then it is also an eigenvector of C_s associated with eigenvalue $\lambda_s = \lambda_x - \alpha$.

$$\begin{aligned}
C_x \boldsymbol{u} &= \lambda_x \boldsymbol{u} = C_s \boldsymbol{u} + \alpha I \boldsymbol{u} \\
C_s \boldsymbol{u} &= \lambda_x \boldsymbol{u} - \alpha I \boldsymbol{u} \\
C_s \boldsymbol{u} &= (\lambda_x - \alpha) \boldsymbol{u} \\
\text{but} \quad C_s \boldsymbol{v} &= \lambda_s \boldsymbol{v} \\
\implies \lambda_s &= \lambda_x - \alpha \qquad \text{is an eigenvalue of } C_s \text{ with associated eigenvector } \boldsymbol{u}
\end{aligned}$$

2. Invertibility of a particular symmetric matrix

Let $A \in \mathbb{R}^{m \times n}$. Show that the matrix M defined as

$$M = \alpha^2 I + AA^T, \quad \alpha \neq 0 \in \mathbb{R}$$

is nonsingular, where $I = I_m$ and α is a nonzero real number.

First, we will show that M is positive-definite. For any $x \neq 0 \in \mathbb{R}^m$,

$$\begin{aligned}
x^T M x &= x^T(\alpha^2 I + AA^T)x = \alpha^2 x^T x + x^T AA^T x \\
&= \alpha^2 ||x||^2 + ||A^T x||^2 \\
&\geq \alpha^2 ||x||^2 \qquad \text{if } x \in \text{null}(A^T) > 0 \\
\implies x^T M x &> 0,
\end{aligned}$$

thus, M is positive definite which implies that the eigenvalues of M are positive. Since zero cannot be an eigenvalue, M is nonsingular.

3. Between-Class Scatter Matrix simplification

Show that the between-class scatter matrix, S_B, in the multi-class *Fisher Discriminant Analysis* is given by

$$S_B = \sum_{i=1}^{M} n_i (\boldsymbol{m}_i - \boldsymbol{m})(\boldsymbol{m}_i - \boldsymbol{m})^T,$$

where M is the total number of distinct classes, n_i is the number of data points in class i, $\boldsymbol{m}_i$ is the class mean of the i^{th} class, and m is the mean across all n data points. You may use the facts that

$$S_T = S_B + S_W, \quad S_W = \sum_{i=1}^{M} \sum_{x \in D_i} (x - \boldsymbol{m}_i)(x - \boldsymbol{m}_i)^T, \quad \text{and} \quad S_T = \sum_{i=1}^{n} (x_i - \boldsymbol{m})(x_i - \boldsymbol{m})^T$$

Given what we know above, and the fact that the ith class mean $\boldsymbol{m}_i = \frac{1}{n_i} \sum_{x \in D_i} \boldsymbol{x}$ and the mean $\boldsymbol{m}$ across all data points is given by $\boldsymbol{m} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i$, we will start with evaluating $S_B = S_T - S_W$.

Furthermore, note that $\forall \boldsymbol{x}_i \in X$ (the entire data set), $\boldsymbol{x}_i \in \bigcup_{j=1}^{M} D_j$, the union of all distinct classes, and also $\boldsymbol{m}_i n_i = \sum_{x \in D_i} \boldsymbol{x}$.

$$\begin{aligned}
S_T - S_W &= \sum_{i=1}^{n} (x_i - \boldsymbol{m})(x_i - \boldsymbol{m})^T - \sum_{i=1}^{M} \sum_{x \in D_i} (x - \boldsymbol{m}_i)(x - \boldsymbol{m}_i)^T \\
&= \sum_{i=1}^{M} \sum_{x \in D_i} \left[(x - \boldsymbol{m})(x - \boldsymbol{m})^T - (x - \boldsymbol{m}_i)(x - \boldsymbol{m}_i)^T \right] \\
&= \sum_{i=1}^{M} \sum_{x \in D_i} \left[(xx^T - \boldsymbol{m}x^T - x\boldsymbol{m}^T + \boldsymbol{m}\boldsymbol{m}^T) - (xx^T - \boldsymbol{m}_i x^T - x\boldsymbol{m}_i{}^T + \boldsymbol{m}_i \boldsymbol{m}_i{}^T) \right] \\
&= \sum_{i=1}^{M} \sum_{x \in D_i} \left[(\boldsymbol{m}\boldsymbol{m}^T - \boldsymbol{m}x^T - x\boldsymbol{m}^T) - (\boldsymbol{m}_i \boldsymbol{m}_i{}^T - \boldsymbol{m}_i x^T - x\boldsymbol{m}_i{}^T) \right] \\
&= \sum_{i=1}^{M} \left[n_i \boldsymbol{m}\boldsymbol{m}^T - n_i \boldsymbol{m}_i \boldsymbol{m}_i{}^T + \sum_{x \in D_i} (\boldsymbol{m}_i x^T + x\boldsymbol{m}_i{}^T - \boldsymbol{m}x^T - x\boldsymbol{m}^T) \right] \\
&= \sum_{i=1}^{M} \left[n_i \boldsymbol{m}\boldsymbol{m}^T - n_i \boldsymbol{m}_i \boldsymbol{m}_i{}^T + \boldsymbol{m}_i \sum_{x \in D_i} x^T + \boldsymbol{m}_i{}^T \sum_{x \in D_i} x - \boldsymbol{m} \sum_{x \in D_i} x^T - \boldsymbol{m}^T \sum_{x \in D_i} x \right] \\
&= \sum_{i=1}^{M} \left[n_i \boldsymbol{m}\boldsymbol{m}^T - n_i \boldsymbol{m}_i \boldsymbol{m}_i{}^T + \boldsymbol{m}_i n_i \boldsymbol{m}_i{}^T + \boldsymbol{m}_i{}^T n_i \boldsymbol{m}_i - \boldsymbol{m} n_i \boldsymbol{m}_i{}^T - \boldsymbol{m}^T n_i \boldsymbol{m}_i \right] \\
&= \sum_{i=1}^{M} n_i \left[\boldsymbol{m}\boldsymbol{m}^T + \boldsymbol{m}_i{}^T \boldsymbol{m}_i - \boldsymbol{m}\boldsymbol{m}_i{}^T - \boldsymbol{m}^T \boldsymbol{m}_i \right] \\
&= \sum_{i=1}^{M} n_i (\boldsymbol{m} - \boldsymbol{m}_i)(\boldsymbol{m} - \boldsymbol{m}_i)^T
\end{aligned}$$

Computing

1. KL Procedure for Gappy Data

This project concerns the application of the KL procedure for incomplete data [3]. Let the complete data set be translation-invariant:

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

where $m = 1, \ldots, M$, with M dimension of the ambient space (size of the spatial grid), and $\mu = 1, \ldots, P$, with P the number of points in the ensemble.

Let $x_m = \dfrac{2\pi(m-1)}{M}$ and $t_\mu = \dfrac{2\pi(\mu-1)}{P}$.

Select an ensemble of masks $\{\boldsymbol{m}^{(\mu)}\}$, $\mu = 1, \ldots, P$. where 10% of the indices are selected to be zero for each mask. Each pattern in the incomplete ensemble may be written as

$$\tilde{\boldsymbol{x}}^{(\mu)} = \boldsymbol{m}^{(\mu)}.\boldsymbol{f}^{(\mu)},$$

where $\left(\boldsymbol{f}^{(\mu)}\right)_m = \frac{1}{N}\sum_{k=1}^{N}\frac{1}{k}\sin[k(x_m - t_\mu)]$. Let $P = M = 64$ and $N = 3$.

a. Compute the eigenvectors of this ensemble using the gappy algorithm [3].

b. Plot the eigenvalues as a function of the iteration, and continue until they converge.

c. Plot your final eigenfunctions corresponding to the 10 largest eigenvalues.

d. Plot the element $\tilde{\boldsymbol{x}}^{(1)}$ and the vector $\tilde{\boldsymbol{x}}_D$ repaired according to Equation

$$\tilde{\boldsymbol{x}} \approx \tilde{\boldsymbol{x}}_D = \sum_{n=1}^{D}\tilde{a}_n\phi^{(n)}. \tag{1}$$

Determine the value of D that provides the best approximation to the original non-gappy pattern vector.

We begin by visualizing the true and gappy data (generated using a random mask) shown in Figures 1 and 2, respectively.

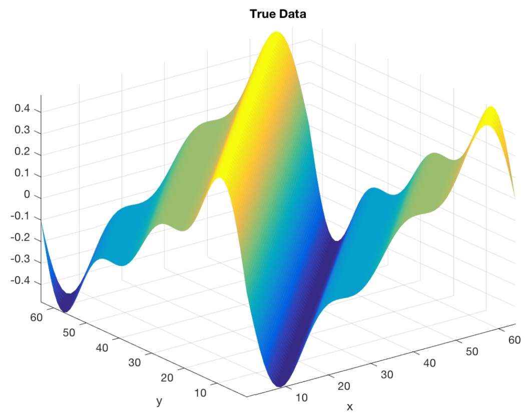

Figure 1: True Data (Generated)

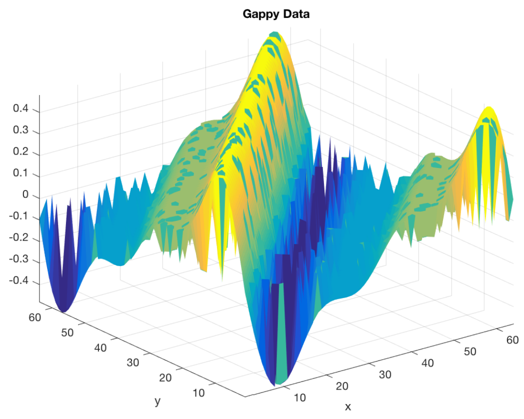

Figure 2: Gappy Data (“Given”)

The procedure for filling in the gappy data is outlined in [1], and the actual code for doing so is given later in this document. Given a set of data $\{\boldsymbol{x}^{(\mu)}\}_{\mu=1}^{P}$ with each $\boldsymbol{x}^{(\mu)} \in \mathbb{R}^{M\times P}$, we initialize masks for each data point $\{\boldsymbol{m}^{(\mu)}\}_{\mu=1}^{P}$ to define our gappy data. If $m_i^{(\mu)} = 0$, there is a gap and we need to repair the data; if $m_i^{(\mu)} = 1$, do nothing. We are changing/repairing only the gaps (defined by the 0-values in a mask vector m) with a *repair* $\boldsymbol{r}$, where

$$\boldsymbol{r} = \sum_{n=1}^{D} b_n U^{(n)}$$

where D is a rank-approximation value, and $\{U^{(n)}\}$ is a set of basis vectors (eigenvectors) of the data set $\{x^{(\mu)}\}_{\mu=1}^{P}$. The coefficients b_n for each set of data are found by solving the system of equations

$$\begin{bmatrix} \langle \boldsymbol{x}^{(\mu)}, U^{(1)}\rangle_m \\ \vdots \\ \langle \boldsymbol{x}^{(\mu)}, U^{(D)}\rangle_m \end{bmatrix} = \begin{bmatrix} \langle U^{(1)}, U^{(1)}\rangle_m & \cdots & \langle U^{(1)}, U^{(D)}\rangle_m \\ \vdots & \ddots & \vdots \\ \langle U^{(D)}, U^{(1)}\rangle_m & \cdots & \langle U^{(D)}, U^{(D)}\rangle_m \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_D \end{bmatrix}$$

where $\langle U^{(i)}, U^{(j)}\rangle_{\boldsymbol{m}}$ is the dot product of $U^{(i)}, U^{(j)}$ with the applied mask $\boldsymbol{m}$. In other words,

$$\langle U^{(i)}, U^{(j)}\rangle_{\boldsymbol{m}} = \begin{bmatrix} U_1^{(i)} m_1 & U_2^{(i)} m_2 & \cdots & U_M^{(i)} m_M \end{bmatrix} \begin{bmatrix} U_1^{(j)} m_1 \\ U_2^{(j)} m_2 \\ \vdots \\ U_M^{(j)} m_M \end{bmatrix}$$

Solving for the repair $\boldsymbol{r}$ for each $\boldsymbol{x}^{(\mu)}$ gives us an approximation to the true data set, and we iterate this algorithm until convergence. With that, let's look at the results for the data set shown in Figure 1. We use the first iteration of this algorithm to determine an appropriate D-approximation to the data set.

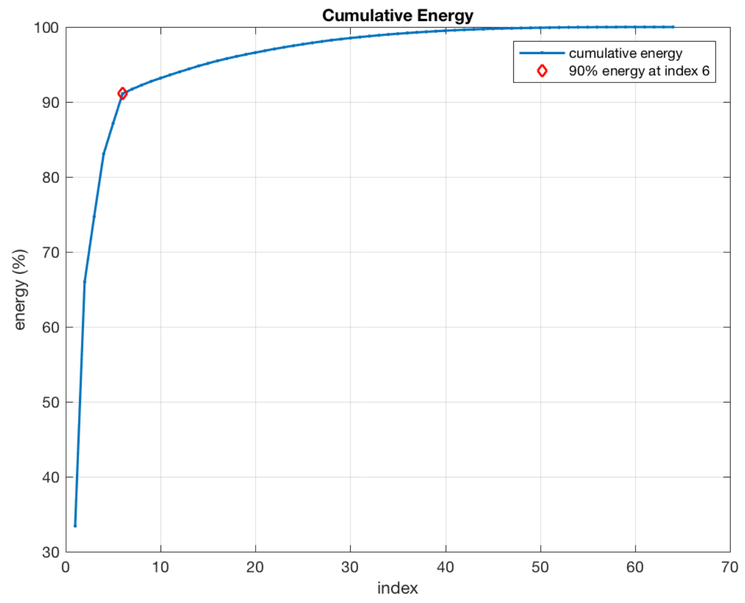

Figure 3: Cumulative energy of the mean-subtracted gappy data

From Figure 3, we select a $D = 6$ rank approximation of the data to retain 90% of the energy. So for the remaining iterations, we will use the rank-6 approximation when obtaining the best basis (SVD). To further illustrate this point, we plot the first gappy data point along with its repair, i.e. estimation of the gappy data using the coefficients b_n from above:

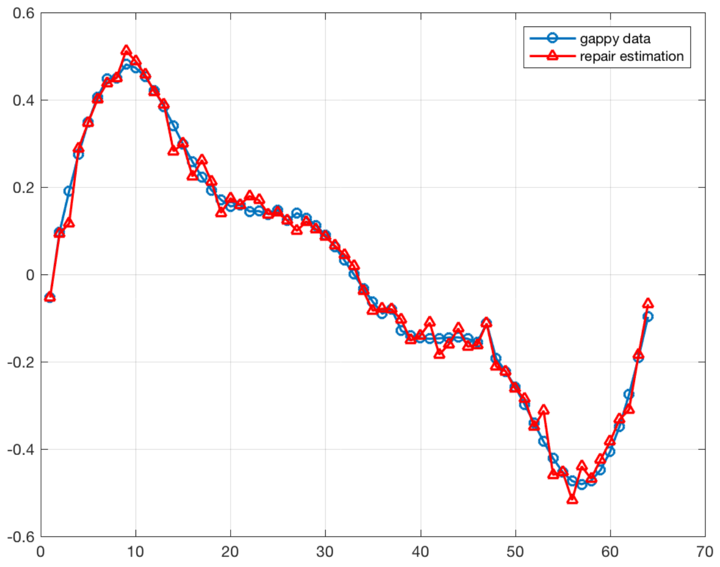

Figure 4: First iteration repair

Using this rank-6 approximation, we iterate until convergence using the eigenvalues of the repaired data as a critera: when the maximum absolute change in eigenvalues falls below $1e-4$, we consider the algorithm to have converged. With this tolerance, the algorithm converges in 8 iterations. For a tolerance of $1e-8$, we get convergence in 18 iterations. The convergence of the eigenvalues is shown in Figure 5.

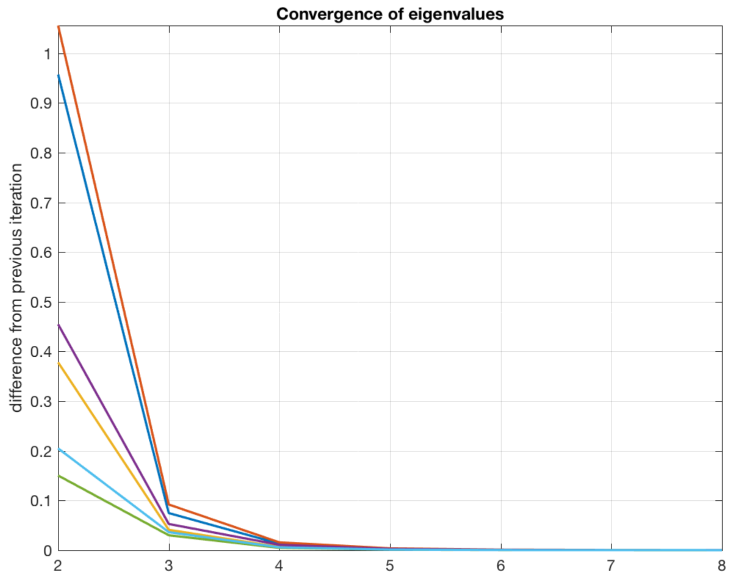

Figure 5: Eigenvalue convergence

The convergence of the data is shown below with the error from the true data. Depending on the application, the repair given in iteration 3 is visually very similar to the true data from Figure 1.

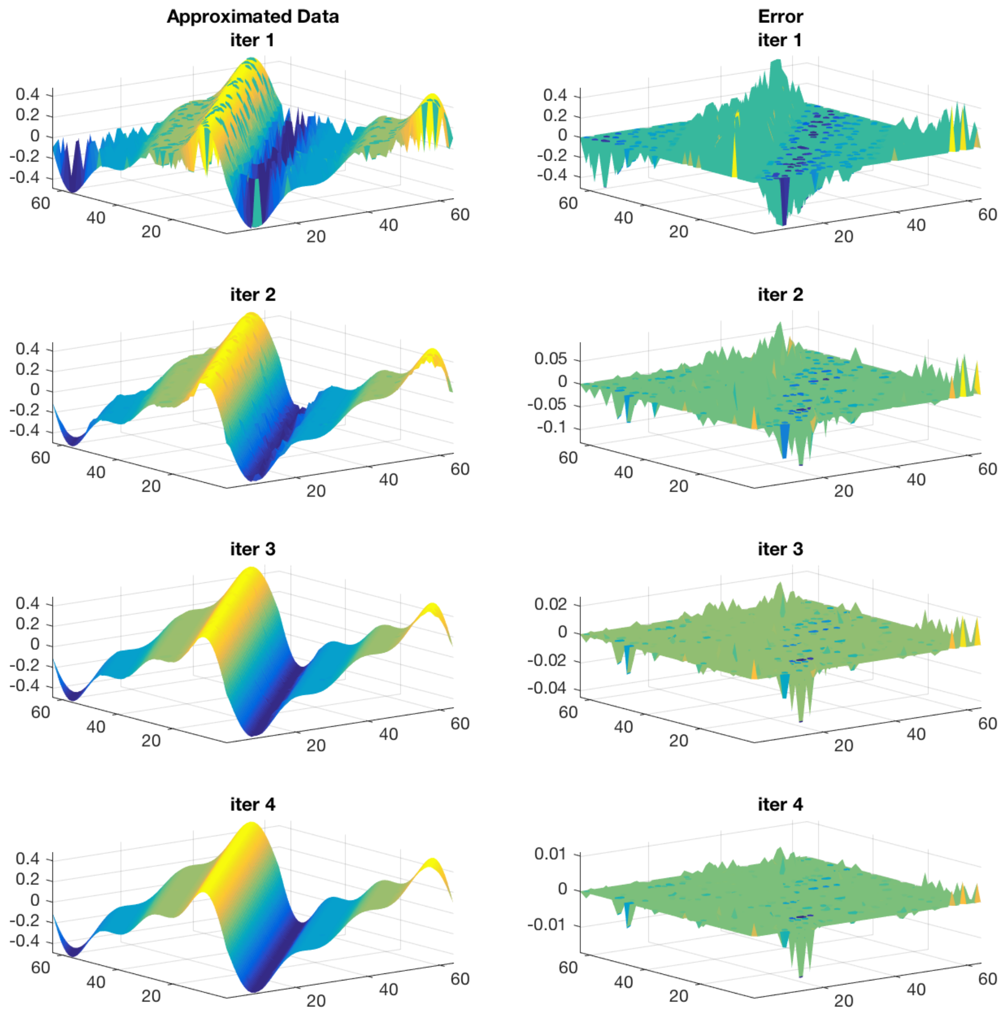

Figure 6: Data convergence for the first 4 iterations

Let us also examine the first ten eigenvectors of the repaired set. As expected, the eigenvectors / eigenfunctions of this data are sinusoidal. Further eigenvectors do not show anything intuitive as they are associate with low singular values. For reference, the first several singular values are given as: $\{\frac{32}{3}, \frac{32}{3}, \frac{16}{3}, \frac{16}{3}, \frac{32}{9}, \frac{32}{9}, 0.0006, 0.0002, 0.0002, 0.0001, 0.0001, 0.0001\}$. As we can see, the first 6 eigenvalue-eigenvector pairs make up the bulk of the data.

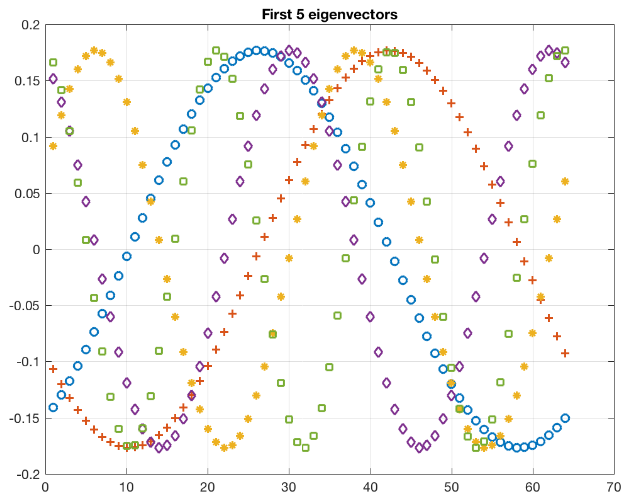

Figure 7: Eigenvectors 1 to 5

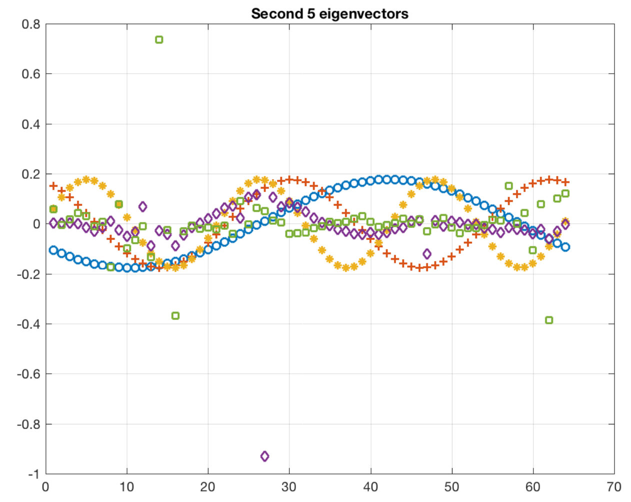

Figure 8: Eigenvectors 6 to 10

2. Linear Discriminant Analysis (LDA)

a. Write a MATLAB routine to produce an optimal projection direction, w, using the two-class LDA criterion

$$w = \arg\max_w J(w) = \arg\max_w \frac{w^T S_B w}{w^T S_W w},$$

where

$$S_B = (\boldsymbol{m}_2 - \boldsymbol{m}_1)(\boldsymbol{m}_2 - \boldsymbol{m}_1)^T \quad \text{and} \quad S_W = \sum_{i=1}^{M} \sum_{x \in D_i} (x - \boldsymbol{m}_i)(x - \boldsymbol{m}_i)^T$$

are the between-class scatter matrix and the within-class scatter matrix, respectively. That is, your code should take in a set of data points with a clear indication which points belong to class 1 and which points belong to class 2, and output a single vector w that is the solution of the generalized eigenvalue problem $S_B w = \lambda S_W w$.

b. Now, use your subroutine in part (a) to project the EEG data onto a real line. Particularly, we can form a data point in $\mathbb{R}^{1040\times 19}$ by concatenating the columns for each trial, therefore having 10 data points for task 2 and 10 data points for task 3. You would then project these 20 points onto the real line with the w found with part (a). Plot the projected data on the real line and distinguish the classes with different symbols. Do you see a clear separation? Analyze your results.

For the algorithm to make sense, we need to recall some important properties about the SVD. A data set (matrix) $X \in \mathbb{R}^{m\times p}$ ($m > p$, $\text{rank}(X) = p$) can be decomposed into the "thin" SVD:

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

where U is $m \times p$, S is $p \times p$, and V is $p \times p$. In our particular data, we have X as an 19760×20 matrix, and it is not feasible to find the S_B and S_W matrices due to limited computer memory. If we instead perform LDA on $\tilde{X} = \Sigma V^T \in \mathbb{R}^{p\times p}$, we can use the projection vector w on $\tilde{X}$ to classify the data. The code is include later in this document, but the general outline is this:

- Perform thin SVD on X: $[U, S, V] = \text{svd}(X, 0)$
- Apply PCA to obtain a D-rank approximation. We now have $\hat{U}, \hat{S}, \hat{V}$ that have D columns each (instead of p columns).
- Form a modified data set, $A = \hat{S}\hat{V}^T$ [5]
- Formulate the between-class scatter matrix, S_B, and the within-class scatter matrix, S_W, based on A
- Get the eigenvector w associated with the maximum eigenvalue of $S_W^{-1} S_B$
- Project: $w^T A$. This should give the desired classes.

The results for the EEG data shows clear separation, along with an interesting variation depending on how we calculate the SVD. In particular, we find that the separation is very clear when using just the SVD on X to reduce the data (Figure 9). If we instead use SVD on $X - \langle X \rangle$, the mean-subtracted data, we obtain the results shown in Figure 10. There is still a clear separation between classes, and the data points are classified correctly, but the projected points have more variance in the w space.

As a quick guess, this may happen because the raw data has very strong variance between the classes. Performing SVD on the mean-subtracted *entire* data set, we are normalizing both classes to be "closer" together.

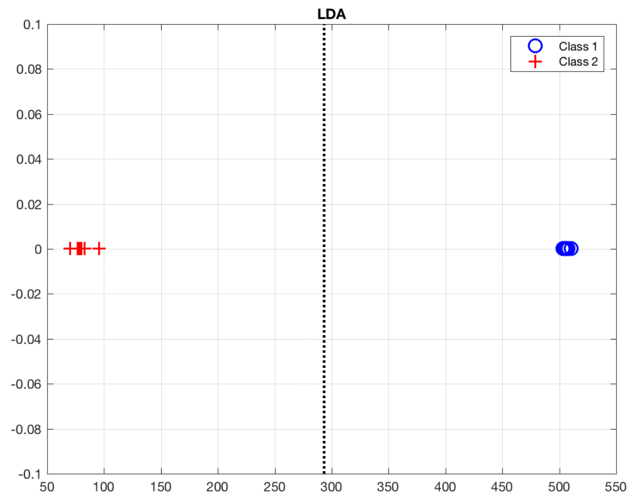

Figure 9: LDA using SVD directly

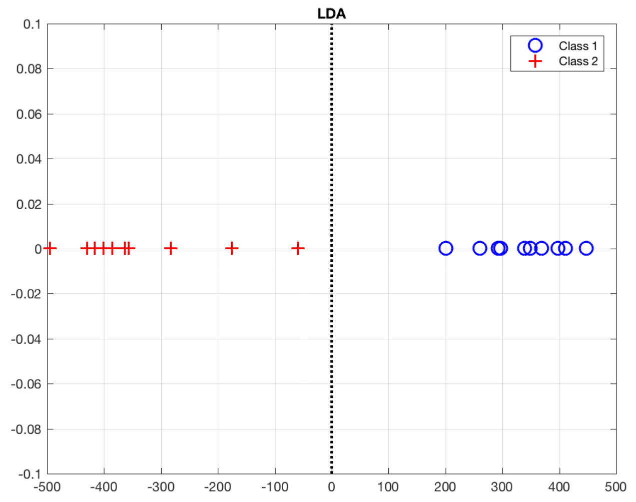

Figure 10: LDA using SVD on the mean-subtracted data

3. Maximum Noise Fraction (MNF) method

Construct a $n \times 10$ matrix (choose $n \geq 250$) to serve as a ground truth data set so that each column is a n-dimensional time series. Next, add **correlated noise** to each column to create a noisy data set, X. The goal of this problem is to implement the Maximum Noise Fraction method to recover the ground truth as closely as possible from the noisy data. Suppose the source of the noise is unknown, you may estimate the noise covariance, $N^T N$, using the difference matrix as $N^T N = \frac{1}{2} dX^T dX$ where if

$$X = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \cdots & x_p(t_1) \\ x_1(t_2) & x_2(t_2) & \cdots & x_p(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_n) & x_2(t_n) & \cdots & x_p(t_n) \end{bmatrix}$$

then

$$dX = \begin{bmatrix} x_1(t_2) - x_1(t_1) & x_2(t_2) - x_2(t_1) & \cdots & x_2(t_2) - x_p(t_1) \\ x_1(t_3) - x_1(t_2) & x_2(t_3) - x_2(t_2) & \cdots & x_2(t_3) - x_p(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_n) - x_1(t_{n-1}) & x_2(t_n) - x_2(t_{n-1}) & \cdots & x_2(t_n) - x_p(t_{n-1}) \end{bmatrix}$$

Notice that $X \in \mathbb{R}^{n \times p}$ and $dX \in \mathbb{R}^{(n-1) \times p}$. In your report, examine and elaborate on the effect of a D-mode reconstruction on a single noisy signal for various values of D (i.e., choose a single column to filter). In a single graph, visually display the result of the original signal, noisy signal, and filtered (de-noised) data (with your best choice of D) to compare. Use the graph legend to distinguish each.

According to [4], the MNF optimal basis is found via:

- Take the eigenvector expansion of the covariance of $dX^T dX = V_1 \Sigma_1^2 V_1^T$.
- Whiten the original data: $\hat{X} = X V_1 \Sigma_1^{-1}$.
- Use the eigenvector expansion of the covariance of $\hat{X}$: $\hat{X}^T \hat{X} = U_1 \Sigma_2^2 U_1^T$.
- Define $\Psi = V_1 \Sigma_1^{-1} U_1^T$.
- Compute the maximum noise fraction basis vectors via $\Phi = X\Psi$.

Then our signal $S = \Phi$. Unfortunately, I was not able to effectively produce desired results, as seen in Figure 11.

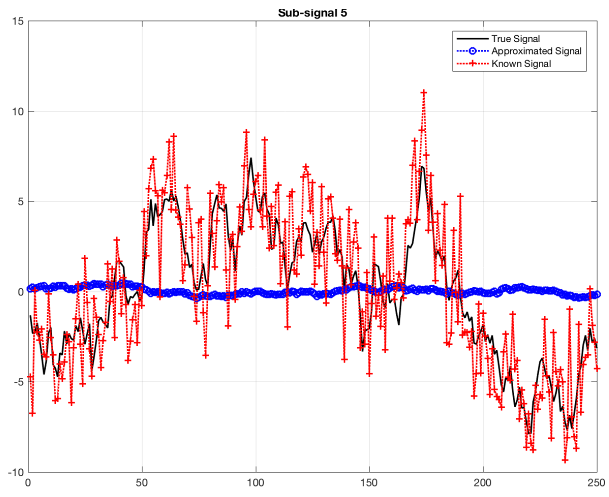

Figure 11: MNF for noise elimination

Code

KL Procedure for Gappy Data

```
function [xhat, eigData] = repair_gappy_data(xhat, mask, maxit)
% REPAIR_GAPPY_DATA
%
% Syntax:
%   [X_repair, eigData] = REPAIR_GAPPY_DATA(X, mask)
%
% where X is the given data and mask is the matrix indicating where the
% gaps are to be filled in (by value=0 at gap, value=1 elsewhere).
%
% The outputs are the repaired data, X_repair, and the iteration data
% containing the eigenvalues and eigenvectors in a cell:
%       eigData = {U_0, sigma_0;
%                  U_1, sigma_1;
%                   ...
%                  U_iter, sigma_iter};
%
% Algorithm: assuming we don't have a good basis --
%
%   1) make an initial repair with the ensemble average
%   2) compute first estimate of KL basis
%   3) do until convergence:
%         a) re-estimate the gappy data using Type 1 algorithm
%         b) recompute the KL basis
%

% Author: Kristin Holmbeck

if nargin == 2
    maxit = 50;      % maximum number of iterations
end

cvg_tol      = 1e-6;
P            = size(xhat,2);
Pi           = sum(mask,2);
xavg         = sum(xhat,2) ./ Pi;
xavg         = repmat(xavg, 1, P);
xhat(~mask) = xavg(~mask);          % fill in initial gaps with ensemble avg

eigData = {};
D       = P;

for iter = 1:maxit
    [U,S,V] = best_basis(xhat);     % KL-basis

    if iter == 1
        E = cumulative_energy(diag(S), rank(xhat));
        D = find(E>0.90, 1);     % use 90% rank approximation
    end

    % need to do a D-approximation of xhat
    U = U(:,1:D);
    S = S(1:D,1:D);

    eigData{iter,1} = U;
    eigData{iter,2} = diag(S);
    eigData{iter,3} = xhat;

    if iter > 1
        sigdiff = eigData{iter,2} - eigData{iter-1,2};
        if max(abs(sigdiff)) < cvg_tol
            return
        end
    end

    b = zeros(D,P);

    for mm = 1:P

        mu      = mask(:,mm);
        Umask   = U .* repmat(mu, 1, D);
        Ms      = Umask' * Umask;
        f       = Umask' * (xhat(:,mm).*mu);

    %{
        % The long way of doing it --
        Ms = zeros(D,D);
        for ii = 1:size(Ms,1)
            for jj=1:size(Ms,2)
                Ui = U(:,ii);
                Uj = U(:,jj);
                Ms(ii,jj) = (Ui.*mu)' * (Uj.*mu);
            end
        end
        f = zeros(D,1);
        for jj = 1:D
            f(jj) = (xhat(:,mm).*mu)' * Umask(:,jj);
        end
    %}

        b(:,mm) = pinv(Ms)*f;
    end

    repair      = U*b;
    xhat(~mask) = repair(~mask);
end

end
```

LDA

```
function [w,Xproj,alpha] = LDA(DATA, classes)
% LDA
% Linear Discriminant Analysis
%
% Syntax:
%   w = LDA(X, class)
%
% where X is the entire data set and class indicates the class of each
% column of X. The function returns the projection vector, w, a vector
% whose length equals the number of columns of X, and the projected data,
% Xproj.
%

% Author: Kristin Holmbeck

allClasses  = unique(classes);
```

```
nClasses     = length( allClasses );
ni           = zeros(nClasses,1);
for ii = 1:nClasses
    ni(ii) = sum(classes == allClasses(ii));
end

% approximate the data with lower dimensions (PCA)
% and transform into a new space (X)
[U,S,V] = svd(DATA,0);
% [U,S,V] = best_basis(DATA);
E        = cumulative_energy(diag(S), rank(DATA));
k        = find(E>0.95, 1);          % 95% rank approximation
X        = S(1:k,1:k)*V(:,1:k)';     % use this new space

nDat     = size(X,2);
m        = sum(X,2) / nDat;          % total mean

SB = zeros(k);
SW = zeros(k);
for ii = 1:nClasses
    thisClass   = ( classes==allClasses(ii) );
    classData   = X(:,thisClass);
    mu          = sum(classData,2) / ni(ii);   % current classwise mean
    SB          = SB + ni(ii)*(m-mu)*(m-mu)';

    for jj = 1:size(classData,2)
        SW = SW + (classData(:,jj) - mu)*(classData(:,jj) - mu)';
    end
end

A        = pinv(SW)*SB;
[V,D]    = eig(A);
D        = abs(diag(D));
[D,ndx] = max(D);
w        = V(:,ndx);
Xproj    = w'*X;
alpha    = w'*m;         % alpha is only valid for nClasses=2

end
```

MNF

```
function [Phi, Basis] = MNF(X,N)
% this algorithm requires the knowledge of N (noise)
% treat N as noise and X as data in the gsvd.

if nargin == 2
    [U,V,A,C,S] = gsvd(N,X,0);
    psi = pinv(A');
    Phi = X*psi; % optimal basis vectors
    Basis = psi;
else
    [Phi,Basis] = MaximumNoiseFraction(X);
end

end

function [Phi,Basis] = MaximumNoiseFraction(X)
% https://www.mathworks.com/matlabcentral/fileexchange/49884-maximumnoisefraction
%
% Function to  compute the Maximum Noise Fraction from a set of signals X(:,i)
% based on paper A SOLUTION PROCEDURE FOR BLIND SIGNAL SEPARATION
% USING THE MAXIMUM NOISE FRACTION APPROACH: ALGORITHMS AND EXAMPLES
%
% Input   X: data set X with rows > column, here each colum is a signal
% Output  BasisVectors: output signals
%         Phi: Transformation matrix


[m, n] = size(X);

% 1. Estimate the covariance of the noise.
dX = zeros(m,n);
for i=1:(m-1)
    dX(i,:) = X(i,:) - X(i+1,:);
end

% Take the eigenvector expansion of the covariance of dX
[U1,S1,V1] = svd(dX'*dX);

% Whiten the original data
wX = X*U1*inv(sqrt(S1));

% Compute the eigenvector expansion of the covariance of wX
[U2,S2,V2] = svd(wX'*wX);

% Define Phi
Phi = U1*inv(sqrt(S1))*U2;

% Compute the Maximum noise fraction basis vectors
Basis = X*Phi;

end
```

References

[1] Chang, Jen-Mei. *Matrix Methods for Geometric Data Analysis and Recognition.* 2014.

[2] P. N. Belhumeur, J. P. Hespanha and D. J. Kriegman, "Eigenfaces vs. Fisherfaces: recognition using class specific linear projection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 711-720, Jul 1997.

[3] R. Everson and L. Sirovich. The karhunen-loeve transform for incomplete data. *J. Opt. Soc. Am., A*, 12(8):16571664, 1995.

[4] Hundley, D., Kirby, M., and Anderle, M. *A SOLUTION PROCEDURE FOR BLIND SIGNAL SEPARATION USING THE MAXIMUM NOISE FRACTION APPROACH: ALGORITHMS AND EXAMPLES.* ¡https://inc.ucsd.edu/ica2001/115-hundley.pdf¿.

[5] Tonne, Debbie. Helped with emphasizing dimensionality reduction before doing LDA.