ISYE 6740 Fall 2025
Homework 2
(100 points + 5 bonus points)

William Pickard

Provided Data: Questions marked with GS must be submitted to Gradescope. You must still include all your results and explanations in this PDF, and include your code in your submitted zip. Failure to pass all gradescope tests will result in losing 50% of the points for that question.

- (GS) Q3: Order of faces using ISOMAP [isomap.dat, isomap.mat]

- (GS) Q4: Eigenfaces and simple face recognition [yalefaces]

All results that are present only in code/notebooks, but not in your PDF report will not be accepted for points.
Handwritten solutions will not be accepted for any reason
For any code that requires randomness, please set your seed as 6740

1. Conceptual questions [30 points].

1. (5 points) Please prove the first principle component direction v corresponds to the largest eigenvalue of the sample covariance matrix:

$$v = \arg \max_{w:\|w\|\leq 1} \frac{1}{m}\sum_{i=1}^{m}(w^T x^i - w^T \mu)^2.$$

You may use the proof steps in the lecture, but please write them logically and cohesively.

1.1 PCA Proof

This objective function tries to find a vector w that maximizes the variance. To begin, we will manipulate the objective function to a form that can be reduced.

$$\begin{align} \frac{1}{m}\sum_{i=1}^{m}(w^T x^i - w^T \mu)^2 &= \frac{1}{m}\sum_{i=1}^{m}(w^T (x^i - \mu))^2 \tag{1} \\ &= \frac{1}{m}\sum_{i=1}^{m} w^T (x^i - \mu) w^T (x^i - \mu) \tag{2} \end{align}$$

Using inner product property:

$$
\begin{aligned}
a^T b &= b^T a \qquad &(3)\\
&= \frac{1}{m}\sum_{i=1}^{m} w^T (x^i - \mu)(x^i - \mu)^T w \qquad &(4)\\
& \qquad &(5)
\end{aligned}
$$

Factor out w since it is independent of i

$$
\begin{aligned}
&: \qquad &(6)\\
&= w^T \left(\frac{1}{m}\sum_{i=1}^{m}(x^i - \mu)(x^i - \mu)^T\right) w \qquad &(7)
\end{aligned}
$$

Now, we can substitute the given covariance matrix equation:

$$
\begin{aligned}
& \qquad &(8)\\
C &= \frac{1}{m}\sum_{i=1}^{m}(x^i - \mu)(x^i - \mu)^T \qquad &(9)\\
\frac{1}{m}\sum_{i=1}^{m}(w^T x^i - w^T \mu)^2 &= w^T C w \qquad &(10)
\end{aligned}
$$

With this simplified form, we can follow a common strategy for solving constrained optimization problems: the Lagrangian function. The constraint on this problem will be that w cannot exceed 1 in magnitude. This must be done, since w is multiplied by itself in the objective function, and without a constraint, the objective function can be made arbitrarily large without a constraint.

With the Lagrangian function, we will take the derivative and set it to zero.

$$
\begin{aligned}
L(w, \lambda) &= w^T C w + \lambda(1 - ||w||^2) \qquad &(11)\\
\frac{\partial L}{\partial w} &= 0 = \frac{\partial(w^T C w)}{\partial w} + \frac{\partial \lambda}{\partial w} - \frac{\partial(\lambda ||w||^2)}{\partial w} \qquad &(12)
\end{aligned}
$$

Using the formula for a quadratic form derivative [PP08, Eq. 81]: $\frac{\partial(x^T B x)}{dx} = (B + B^T)x$

$$\tag{13}$$

and derivative of the matrix L2 norm [PP08, Eq. 131]: $\frac{\partial ||x||_2^2}{\partial x} = 2x$

$$
\begin{aligned}
& \qquad &(14)\\
&= (C + C^T)x - 2\lambda w \qquad &(15)
\end{aligned}
$$

Considering that the covariance matrix is symmetrical, so $C = C^T$

$$
\begin{aligned}
& \qquad &(16)\\
0 &= 2Cw - 2\lambda w \qquad &(17)\\
Cw &= \lambda w \qquad &(18)
\end{aligned}
$$

At the end of the optimization, we find that it is the same equation as the eigenvalue equation $Av = \lambda v$ with λ being the eigenvalue and w being the eigenvector. Therefore, the optimal solution w is an eigenvector of C. Further, if we input the optimized value λw into the reduced objective function (10), we find that the eigenvector w that maximizes the variance of the projection corresponds to the largest eigenvalue λ.

$$\frac{1}{m}\sum_{i=1}^{m}(w^T x^i - w^T \mu)^2 = w^T C w = w^T(\lambda w) \tag{19}$$

From our constraint, we know $w^T w = 1$

$$v = \arg \max_{w:\|w\|\leq 1} \frac{1}{m}\sum_{i=1}^{m}(w^T x^i - w^T \mu)^2 = \lambda \tag{20}$$

2. (5 points) Based on your answer to the question above, explain how to further find the second and third largest principle component directions.

1.2 PCA Continued

If the first principal component corresponds to the largest eigenvalue, the following components correspond to the following eigenvectors in order of their corresponding eigenvalues' magnitude. These directions are orthogonal to each other, and since they're taken in order of the corresponding largest eigenvalues, each additional eigenvector maximizes the remaining variance that has not already been projected.

3. (5 points) Consider the diagonal matrix $A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$. Find two distinct eigenvalue decompositions of A, and prove mathematically that both are valid. This demonstrates that eigendecomposition is not unique. **Note:** Your eigenvalue decompositions must be done mathematically, not programatically.

1.3 Eigendecomposition

Find the eigenvalues:

$$\begin{aligned} det(A - \lambda I) = \det \begin{bmatrix} 3-\lambda & 0 \\ 0 & 2-\lambda \end{bmatrix} &= 0 \\ (3-\lambda)(2-\lambda) &= 0 \\ \lambda_1 = 3, \lambda_2 &= 2 \end{aligned}$$

Find the eigenvectors:

For λ_1:

$$(A - 3I)v = \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} v = 0$$

$$v_1 \text{is free, } v_2 = 0$$

$$v_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

For λ_2:

$$(A - 2I)v = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} v = 0$$

$$v_1 = 0,\ v_2 \text{ is free}$$

$$v_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Prove the first decomposition using $A = P^1 D^1 P^{-1}$. P^1 is the matrix of eigenvectors, D^1 is the diagonal matrix of eigenvalues.

$$A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}, \quad P^1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad D^1 = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} \frac{1}{1} & 0 \\ 0 & \frac{1}{1} \end{bmatrix}$$

$$\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} \frac{1}{1} & 0 \\ 0 & \frac{1}{1} \end{bmatrix}$$

$$\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$$

Prove the second decomposition using $A = P^1 D^1 P^{-1}$. D^1 is the diagonal matrix of eigenvalues. This time, we will scale the eigenvectors in P^1. I'll scale v^1 by 5 and v^2 by 4.

$$A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}, \quad P^1 = \begin{bmatrix} 5 & 0 \\ 0 & 4 \end{bmatrix}, \quad D^1 = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} \frac{1}{5} & 0 \\ 0 & \frac{1}{4} \end{bmatrix}$$

$$\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 15 & 0 \\ 0 & 8 \end{bmatrix} \begin{bmatrix} \frac{1}{5} & 0 \\ 0 & \frac{1}{4} \end{bmatrix}$$

$$\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$$

Eigenvectors are not unique because they are directional components of an eigendecomposition. Therefore, we can scale them by any non-zero scalar and they will still be valid indicators of the direction of the projection.

4. (5 points) Explain the three key ideas in ISOMAP (for manifold learning and non-linear dimensionality reduction).

1.4 ISOMAP Key Ideas

ISOMAP is a non-linear dimensionality reduction technique, unlike PCA which is linear. Standard euclidean distance cannot be used for all data topologies - a classic example in ISOMAP being a swiss roll shape. The following steps of ISOMAP are used to represent curved manifold geodesic distances in lower dimensions.

(a) **Construct a neighborhood graph:** To begin, an adjacency matrix is created that stores information about each node's neighborhood. This can be done by finding each node's k-nearest neighbors or by imposing a static euclidean radius around the node. If two nodes are neighbors, the corresponding weight is the euclidean distance between the nodes. If the nodes fall out of the k-nearest or radius ranges, the weight is zero. This adjacency matrix preserves some information about the manifold structure.

(b) **Compute shortest path distances:** Next, a geodesic distance approximation matrix is created by using the adjacency matrix. For every combination of nodes, we compute the minimum sum of walked distances between neighboring nodes to connect the two nodes. The fact that only neighboring nodes can be walked is the critical piece that approximates the original manifold structure. The resultant matrix preserves the manifold's approximate geodisic distance between any two points - not just neighboring points as in the first idea. As for how the minimum distance is found, there are algorithms such as Floyd-Warshall and Dijkstra's that are classic examples of finding the shortest path between two points in a graph.

(c) **Construct a low-dimensional embedding with MDS:** Finally, multi-dimensional scaling is used to reduce the dimensionality of the geodesic matrix. The MDS attempts to preserve the distances in the matrix while reconstructing to a lower dimension. To do this, classical MDS in ISOMAP converts the distance matrix to a Gram matrix and performs an eigen-decomposition. The top eigenvectors and eigenvalues (up to desired dimensionality) are used to reconstruct the points to a lower dimension.

5. (5 points) Explain how to decide k, the number of principle components, from data.

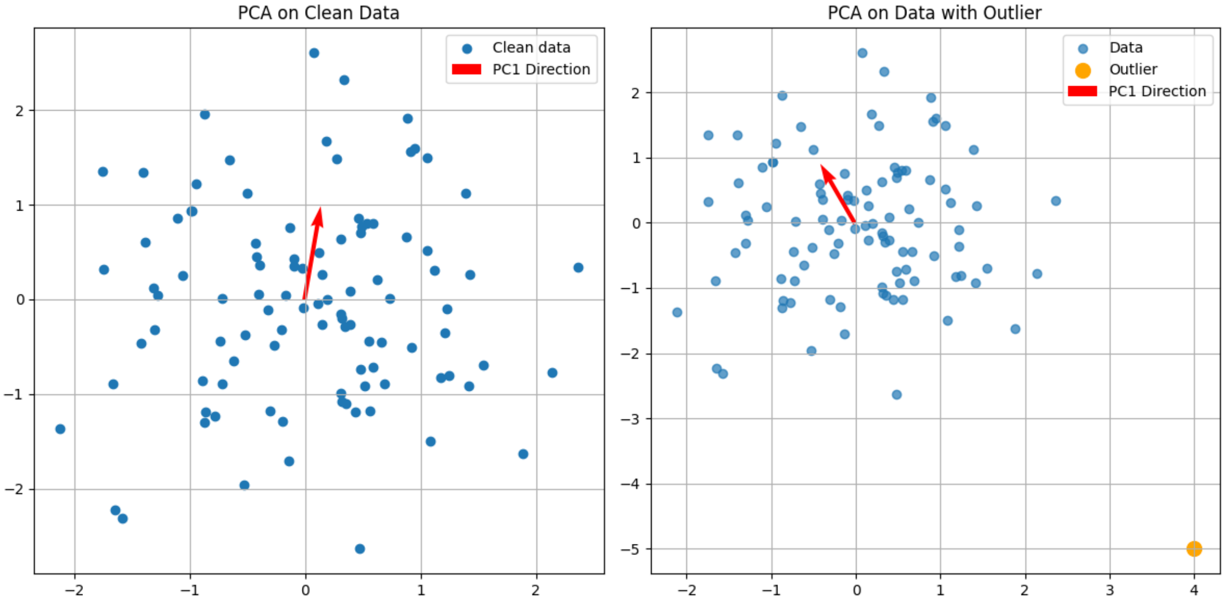
1.5 Choosing k in PCA

In PCA, the point is to reduce dimensionality while preserving as much variance as possible. Therefore, we can decide how many principal components to keep by evaluating how much variance the components account for. The metric for this is called the ”Explained Variance,” and the explained variance per component is $\frac{\lambda_i}{\sum_j=1^n\lambda_j}$. One popular way to use this is to plot each explained variance ratio in order of smallest to largest and assess for an elbow in the graph. This elbow indicates that the remaining components are adding marginally less variance and could be considered for removal. Another way is to set a threshold for cumulative explained variance, where you take however many components are needed to exceed the set threshold (e.g. 95%).

6. (5 points) How do outliers affect the performance of PCA? You can create numerical examples to study and show this.

1.6 PCA and Outliers

The objective function for PCA attempts to maximize the variance of the projected data. Outliers are data points far from the mean of data, meaning they have a large variance that can skew the projection in the direction of an outlier. To demonstrate this, I created a randomized dataset. I copied that dataset and added a significant outlier to the data. After performing PCA on both sets, I plotted the data and the direction of the first principal component in each case. The first principal component on the outlier dataset is significantly skewed toward the outlier.


PCA on Clean Data
Clean data
PC1 Direction
2
1
0
−1
−2
−2
−1
0
1
2
PCA on Data with Outlier
Data
Outlier
PC1 Direction
2
1
0
−1
−2
−3
−4
−5
−2
−1
0
1
2
3
4

2. Graph Laplacian and GNN [20 points].

This question is designed to show how Graph Laplacian and the principle of spectral clustering is used in the context of deep learning, in particular, in developing graph neural networks (GNN). In GNN, such as ChebNet (one of the most popular GNNs), filters are defined as *polynomials of the graph Laplacian*, to extract graph features. This question is meant to build the understanding of how properties from Graph Laplacian can be used in GNN. You can refer to the paper referenced below for additional context.
Ref: M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. NeurIPS, 2016.

Formally, if L is the graph Laplacian and $T_k(\cdot)$ is the Chebyshev polynomial of order k, then

$$g_\theta(L)\,x \;\approx\; \sum_{k=0}^{K} \theta_k\, T_k(\tilde{L})\, x, \qquad \tilde{L} = \tfrac{2}{\lambda_{\max}} L - I.$$

where $\lambda_{\max}$ is the largest eigenvalue of the Graph Laplacian. Here, "a polynomial applied to the graph Laplacian" means taking powers of L (e.g., $I, L, L^2, \dots$) and combining them with learnable coefficients θ_k. Recall from lecture that $L = D - A$, where D is the degree matrix and A is the adjacency matrix.

Example (Chebyshev, small K). Take $K = 2$ and use $T_0(x) = 1$, $T_1(x) = x$, $T_2(x) = 2x^2 - 1$. Then

$$g_\theta(L)\,x \;\approx\; \theta_0\, T_0(\tilde{L})\,x \;+\; \theta_1\, T_1(\tilde{L})\,x \;+\; \theta_2\, T_2(\tilde{L})\,x.$$

For a toy 3-node path with

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad L = D - A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix},$$

one computes

$$L^2 = \begin{bmatrix} 2 & -3 & 1 \\ -3 & 6 & -3 \\ 1 & -3 & 2 \end{bmatrix}, \qquad T_2(L) = 2L^2 - I = \begin{bmatrix} 3 & -6 & 2 \\ -6 & 11 & -6 \\ 2 & -6 & 3 \end{bmatrix}.$$

Thus, a single ChebNet layer with $K = 2$ applies a learnable combination of $\{x,\ Lx,\ T_2(L)x\}$ to produce updated node features—no eigenvectors required.

How this is used in a GNN layer.

- *Input:* a matrix $X \in \mathbb{R}^{n\times d}$ of node features (one row per node).
- *Operation:* build $\{T_k(\tilde{L})X\}_{k=0}^{K}$ (each mixes information up to k hops), linearly combine them with learnable weights, and pass througha nonlinearity.
- *Output:* new node features $X' \in \mathbb{R}^{n\times d'}$ that aggregate information from multi-hop neighborhoods.

Questions. In this exercise, we will show how GNN is constructed based on the idea of graph Laplacian, the same idea we exploited in spectral cluster.

(1) (10 Points) **Neighborhood aggregation.**

(a) Show that

$$(Lx)_i = d_i x_i - \sum_{j\sim i} x_j,$$

so multiplying by L mixes node i's value with its **1-hop neighbors**.

2.a Neighborhood Aggregation Proof

To start, we can expand on Lx

$$\begin{aligned} Lx &= (D - A)x \\ &= Dx - Ax \end{aligned}$$

Looking at this logically, D is a degree matrix and A is an adjacency matrix. Multiplying a vector by a diagonal matrix just scales each element x_i by the degree in D_{ii}. The same logic can be applied to the adjacency matrix, which contains 1 with 1-hop neighbors, and 0 elsewhere. Therefore, the product with x will sum the values x_i where a 1 is present in the row A_i. These operations together are expressed in $d_i x_i - \sum_{j \sim i} x_j$. To show this, we will perform the Lx multiplication and see they are equal.

$$\begin{aligned} Dx &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ 2x_2 \\ x_3 \end{bmatrix} \\ -Ax &= \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -x_2 \\ -x_1 - x_3 \\ -x_2 \end{bmatrix} \\ Lx &= \begin{bmatrix} x_1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + x_3 \end{bmatrix} \end{aligned}$$

Now to show $d_i x_i$ and $\sum_{j \sim i} x_j$ are equal to the above:

$$\begin{aligned} Dx &= \begin{bmatrix} x_1 \\ 2x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ 2x_2 \\ x_3 \end{bmatrix} \\ \sum_{j \sim i} x_j &= \begin{bmatrix} x_2 \\ x_1 + x_3 \\ x_2 \end{bmatrix} \\ Dx - \sum_{j \sim i} x_j &= \begin{bmatrix} x_1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + x_3 \end{bmatrix} \end{aligned}$$

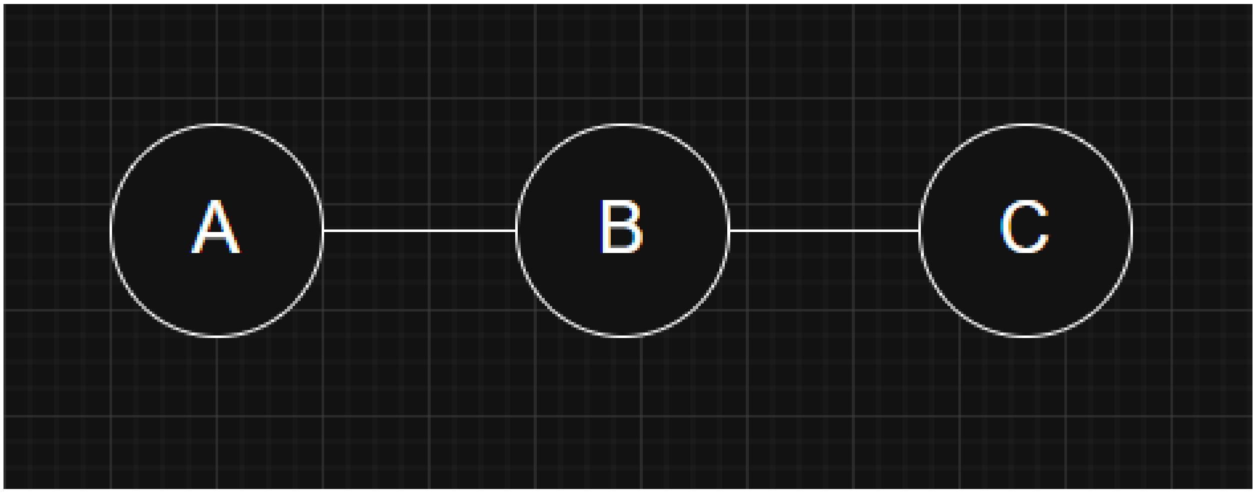
So we can clearly see:

$$(Lx)_i = \begin{bmatrix} x_1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + x_3 \end{bmatrix} = d_i x_i - \sum_{j \sim i} x_j$$

(b) Explain why $L^2 x$ includes terms that traverse **paths of length up to 2** (so information propagates up to 2-hops).

2.b 2 Hop Aggregation

For reference, here is the graph based on the adjacency matrix. This image can be used to help visualize the hopping process.



We will expand L^2 to show that the terms incorporate 2-hop neighbors.

$$L^2x = ((D - A)(D - A))x$$
$$((D - A)(D - A))x = D^2x - DAx - ADx + A^2x$$

Perform the multiplication on each term using the example matrices:

$$= \begin{bmatrix}1 & 0 & 0\\0 & 4 & 0\\0 & 0 & 1\end{bmatrix} x - \begin{bmatrix}0 & 1 & 0\\2 & 0 & 2\\0 & 1 & 0\end{bmatrix} x - \begin{bmatrix}0 & 2 & 0\\1 & 0 & 2\\0 & 2 & 0\end{bmatrix} x + \begin{bmatrix}1 & 0 & 1\\0 & 2 & 0\\1 & 0 & 1\end{bmatrix} x$$

In the above equation, we can see that the D^2x term is scaling by the degree of each node. The $-DAx$ and $-ADx$ terms are dealing with 1-hop neighbors, similar to how we saw in 2.1 with the Lx case. Most interestingly, the A^2x term is the term that incorporates 2-hop neighbors. We can that the following conditions are now non-zero: A connects to C and A, B connects to B, and C connects to C and A. Referencing the graph, we can see that these combinations are all 2-hops. Through these mechanisms, we can see that the L^2x case accounts for path lengths up to 2.

(c) Conclude: why does applying a polynomial in L aggregate information from **multiple hops** in one shot?

2.c Polynomial L Aggregation

From the expansions of the L equation in 2.a and 2.b, we can see that each power of L incorporates information from an additional hop due to the adjacency matrix terms. Therefore, further power of L will continue incorporating more hops as each power of the adjacency matrix will exist in the expanded polynomial up to the maximum number of hops.

Hint

Recall: the off-diagonal entries of L are negative when nodes are connected, and diagonal entries contain the degree. Multiplying by L subtracts neighbor contributions and scales by the degree. Multiplying again by L compounds this mixing, reaching farther neighbors.

(2) (10 Points) **Small example calculation.** Using the 3-node path above, compute $L = D - A$. Then, with $T_0(x) = 1$, $T_1(x) = x$, $T_2(x) = 2x^2 - 1$, write down $T_2(L)$ explicitly.

2.2 Validating given calculation

$$L = D - A = \begin{bmatrix}1 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 1\end{bmatrix} - \begin{bmatrix}1 & -1 & 0\\ -1 & 2 & -1\\ 0 & -1 & 1\end{bmatrix} = \begin{bmatrix}1 & -1 & 0\\ -1 & 2 & -1\\ 0 & -1 & 1\end{bmatrix}$$

$$L^2 = \begin{bmatrix}2 & -3 & 1\\ -3 & 6 & -3\\ 1 & -3 & 2\end{bmatrix}$$

$$T_2(L) = 2L^2 - I = 2\begin{bmatrix}2 & -3 & 1\\ -3 & 6 & -3\\ 1 & -3 & 2\end{bmatrix} - \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}3 & -6 & 2\\ -6 & 11 & -6\\ 2 & -6 & 3\end{bmatrix}$$

This matches the given calculation in the problem.

3. Order of faces using ISOMAP [25 points]

This question aims to reproduce the ISOMAP algorithm results in the original paper for ISOMAP, J.B. Tenenbaum, V. de Silva, and J.C. Langford, Science 290 (2000) 2319-2323 that we have also seen in the lecture as an exercise (isn't this exciting to go through the process of generating results for a high-impact research paper!)

The file isomap.mat (or isomap.dat) contains 698 images, corresponding to different poses of the same face. Each image is given as a 64 × 64 luminosity map, hence represented as a vector in $\mathbb{R}^{4096}$. This vector is stored as a row in the file. (This is one of the datasets used in the original paper.) In this question, you are expected to implement the ISOMAP algorithm by coding it up yourself. You may find the shortest path (required by one step of the algorithm), using `https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csgraph.shortest_path.html`.

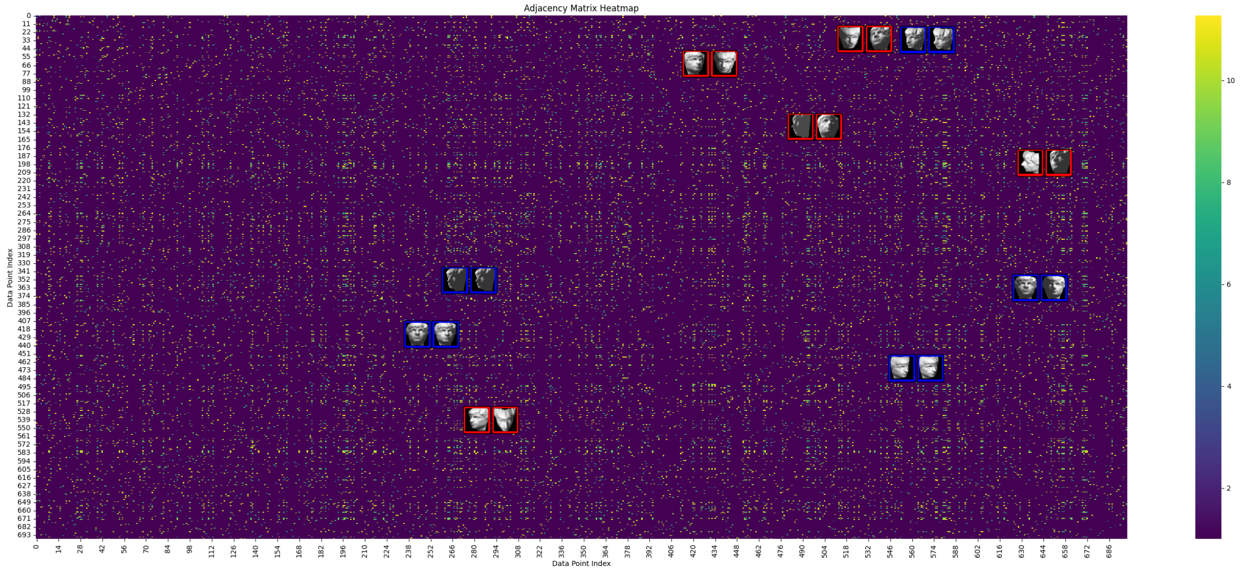
Using Euclidean distance (i.e., in this case, a distance in $\mathbb{R}^{4096}$) to construct the ϵ-ISOMAP (follow the instructions in the slides.) You will tune the ϵ parameter to achieve the most reasonable performance. Please note that this is different from K-ISOMAP, where each node has exactly K nearest neighbors.

1. (5 points) Visualize the nearest neighbor graph through either an image of your adjacency matrix, or visualizing the graph similar to the lecture slides using graph packages such as Gephi (https://gephi.org). Additionally, include a few of the face images that correspond directly to nodes on different parts of your visualization.

3.1 Nearest Neighbor Graph

The first step in creating the nearest neighbor graph is to compute the pairwise distance. Then, we need to find a suitable ϵ to construct the adjacency matrix. The ϵ value serves a radius to determine which nodes are close enough to be considered neighbors. If ϵ is too small, the graph can be entirely disconnected or have suboptimal geodesic distances. If ϵ is too large, the graph manifold structure can be compromised. To find this, I used the scipy `csr_matrix` and `connected_components` functions. `Connected_components` basically determines how many disconnected graphs are present, so I clearly want ϵ to be large enough for only 1 component. From there, I was interested in the average degree of the matrix, since I figured geodesic distance approximation would benefit from each node's degree being larger than 1 - with more potential paths from each node, there are more combinations to find the shortest path. Luckily, the smallest ϵ that connected the graph yielded a large enough average degree, so I chose $\epsilon = 11.27$.

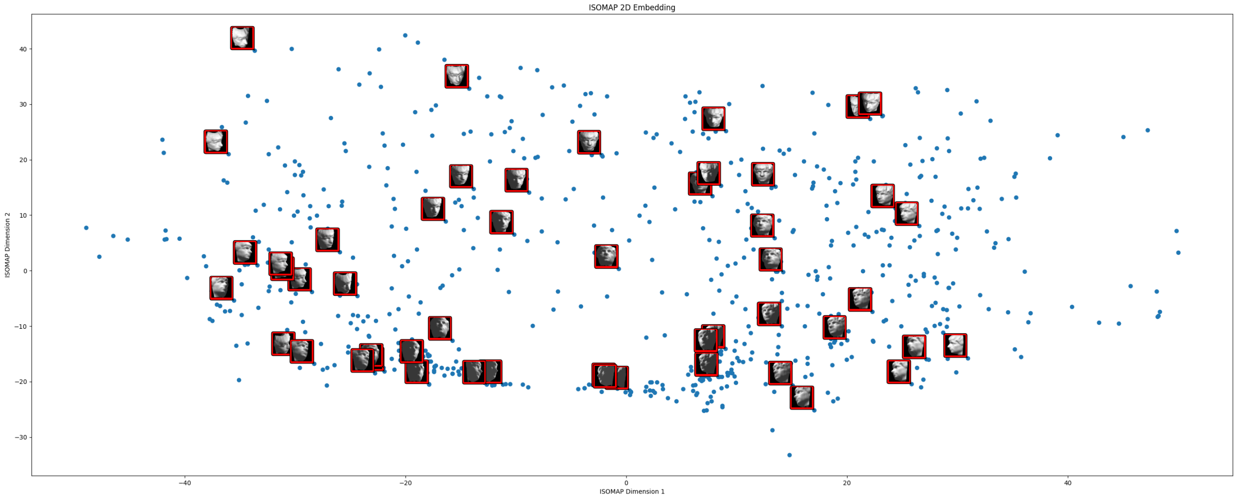
The resultant adjacency matrix is shown below with a heatmap. On the heatmap, purple zones indicate a value of 0, meaning the nodes are not neighbors. The other colors are gradients based on the euclidean distance between the connected nodes. I chose a random set of 10 connected pairs and 10 disconnected pairs and displayed the two images for each pair on the heatmap. The connected pairs are outlined in blue and show clear similarity, proving that the distance metric is working well for neighbors. The disconnected pairs are outlined in red and show clear dissimilarity.



2. (10 points) Implement the ISOMAP algorithm yourself to obtain a two-dimensional low-dimensional embedding. Plot the embeddings using a scatter plot, similar to the plots in lecture slides. Find a few images in the embedding space and show what these images look like and specify the face locations on the scatter plot. Comment on do you see any visual similarity among them and their arrangement, similar to what you seen in the paper? Come up with a way to tune the kernel bandwidth to have desired result.

3.2 ISOMAP 2D Embedding

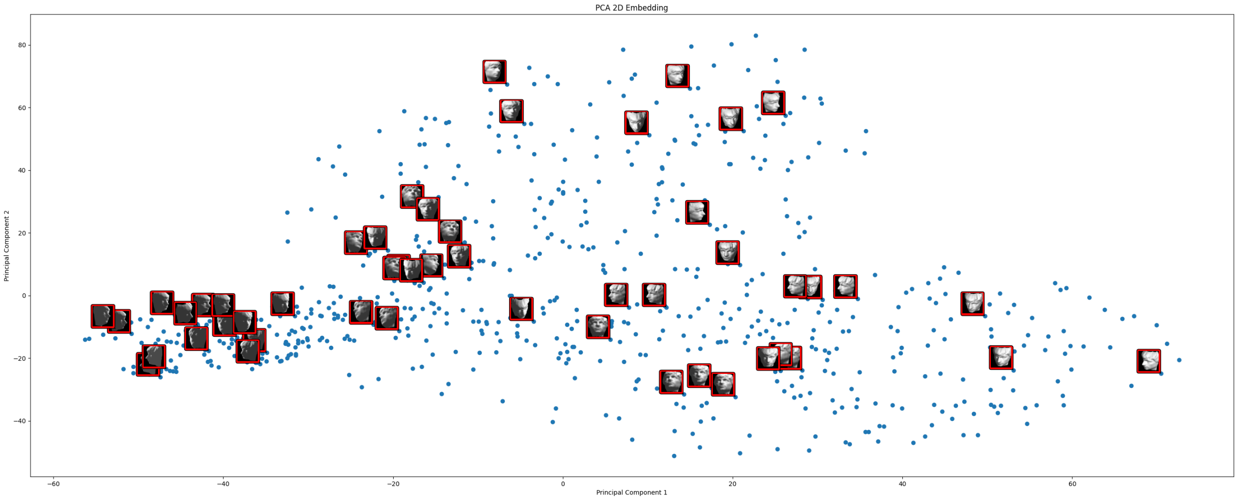
Following the ISOMAP algorithm in the lecture slides, I computed the shortest path distance matrix using the adjacency matrix and the scipy function `shortest_path`. Then, I performed the MDS steps in the slides to extract the 2D embedding. The resultant scatter plot is shown below. The displayed faces for 50 random points are shown and have a clear pattern. Not only are similar faces grouped together, but the face orientations are also ordered in a clear circular pattern. Faces facing left are on the left side of the graph and faces facing right are on the right side of the graph. This matches the original ISOMAP results, which is a good sign to see.



3. (10 points) Perform PCA (you can utilize a PCA package for this) on the images and project them into the top 2 principal components. Again show them on a scatter plot. Explain whether or not you see a more meaningful projection using ISOMAP than PCA.

3.3 PCA 2D Embedding

To compare dimensionality reduction techniques, I performed PCA using sklearn's `StandardScaler` and `pca` to reduce the dimensionality to 2D. The resultant scatter plot is shown below. While the faces are generally grouped together, with similar looking faces near each other, the groupings are clearly not as accurate or distinct as the ISOMAP results. Contributing to this, PCA does not produce a good mapping of the orientation of the faces. Visually, a group of example images along Principal Component 1 = -20 show faces oriented in all sorts of directions. This shows that the lienar PCA technique fails to capture the manifold structure of the data (i.e. the circular orientation pattern of the faces).



4. Eigenfaces and simple face recognition [25 points].

This question is a simplified illustration of using PCA for face recognition. We will use a subset of data from the famous Yale Face dataset.

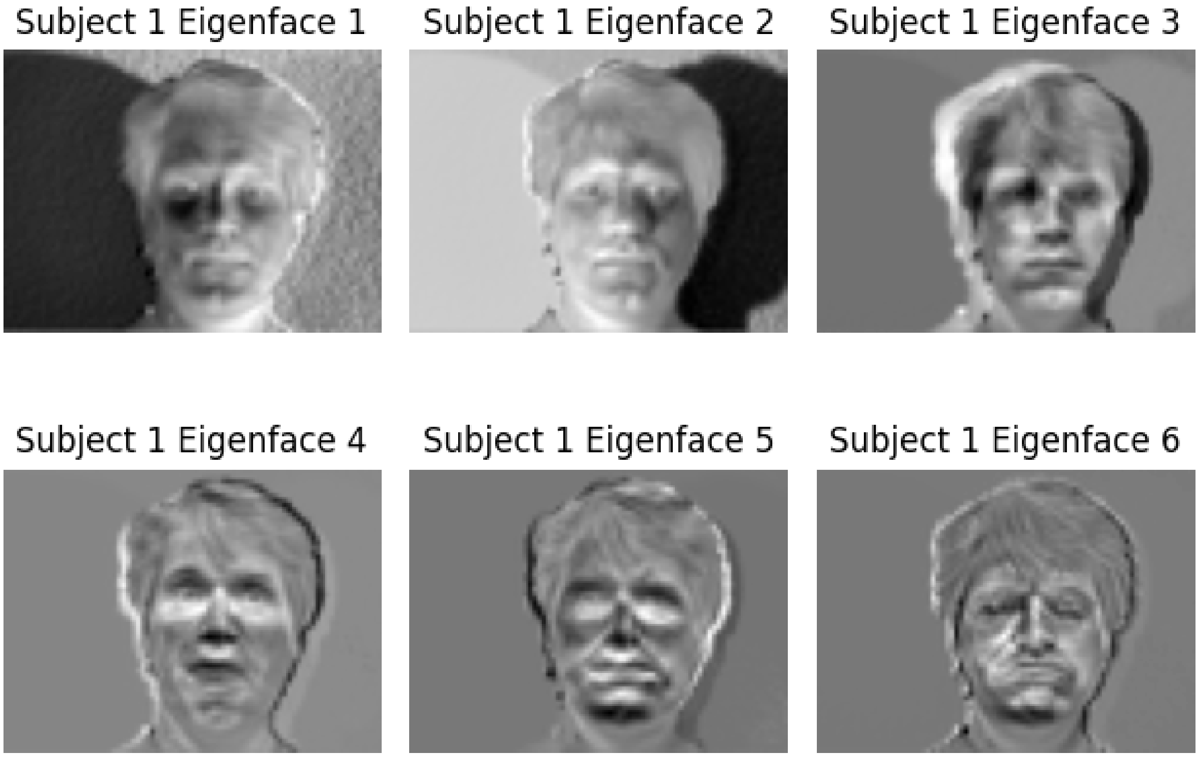
Remark: You will have to perform downsampling of the image by a factor of 4 to turn them into a lower resolution image as a preprocessing (e.g., reduce a picture of size 16-by-16 to 4-by-4). In this question, you can implement your own code or call packages.

First, given a set of images for each person, we generate the eigenface using these images. You will treat one picture from the same person as one data point for that person. Note that you will first vectorize each image, which was originally a matrix. Thus, the data matrix (for each person) is a matrix; each row is a vectorized picture. You will find weight vectors to combine the pictures to extract different "eigenfaces" that correspond to that person's pictures' first few principal components.

1. (10 points) Perform analysis on the Yale face dataset for Subject 1 and Subject 2, respectively, using all the images EXCEPT for the two pictures named subject01-test.gif and subject02-test.gif. **Plot the first 6 eigenfaces for each subject.** When visualizing, please reshape the eigenvectors into proper images. Please explain can you see any patterns in the top 6 eigenfaces?

4.1 Eigenfaces

To start with this, I procedurally load the images, downsample, and flatten them all in the same way. Then, I use sklearn's `StandardScaler` to center the data and Numpy's `SVD` to extract the top 6 principal components. I chose SVD because the matrix was wide, which is a better use for SVD versus PCA. I then plotted the top 6 eigenfaces for each subject. Thes results are shown below.



Subject 1 Eigenface 1

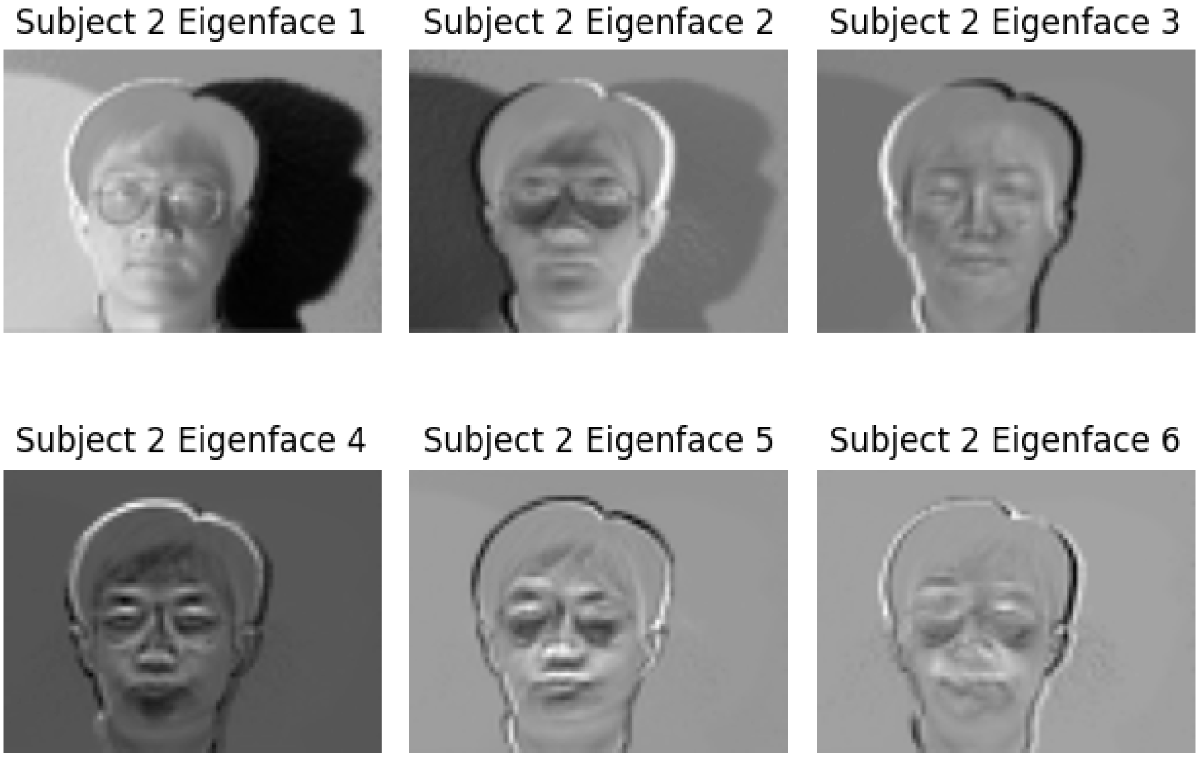
Subject 1 Eigenface 2

Subject 1 Eigenface 3

Subject 1 Eigenface 4

Subject 1 Eigenface 5

Subject 1 Eigenface 6



Subject 2 Eigenface 1

Subject 2 Eigenface 2

Subject 2 Eigenface 3

Subject 2 Eigenface 4

Subject 2 Eigenface 5

Subject 2 Eigenface 6

The eigenfaces seem to get the gist of the subjects' faces. I do notice that the most significant eigenfaces seem to favor any part of the image with hard edges. The gifs with shadows and glasses are very prominent in the top eigenfaces, though there are only 2 training images with shadows and 1 with glasses.

2. (10 points) Now we will perform a simple face recognition task.

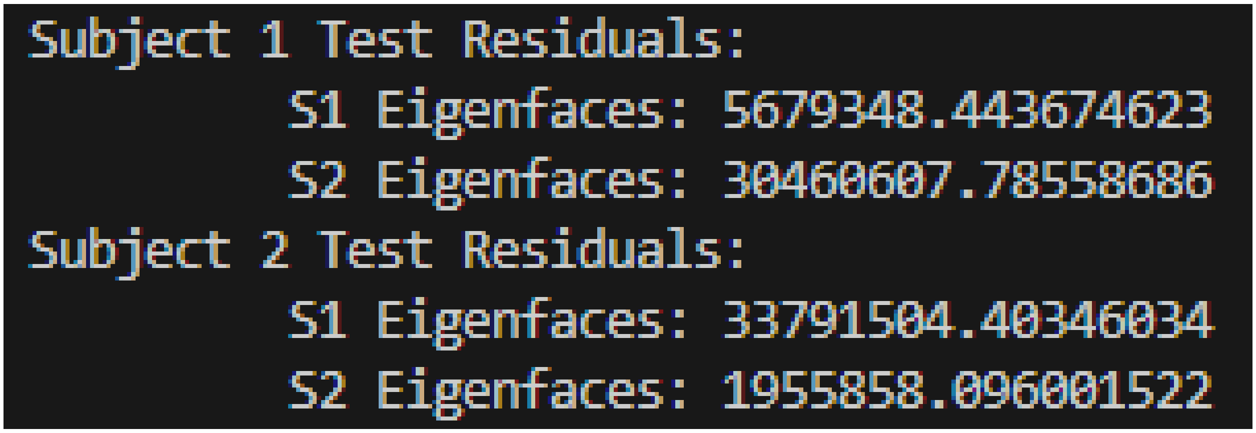
   Face recognition through PCA is proceeded as follows. Given the test image subject01-test.gif and subject02-test.gif, first downsize by a factor of 4 (as before), and vectorize each image. Take the top eigenfaces of Subject 1 and Subject 2, respectively. Then we calculate the *projection residual* of the 2 vectorized test images with the vectorized eigenfaces:

$$s_{ij} = \|(\mathsf{test\ image})_j - (\mathsf{eigenface}_i)(\mathsf{eigenface})_i^T(\mathsf{test\ image})_j\|_2^2$$

   Report all four scores: s_{ij}, $i = 1, 2$, $j = 1, 2$. Explain how to recognize the faces of the test images using these scores.

4.2 Face Recognition

The primary concern in this portion is to ensure that the test image is preprocessed in the same way as the training images. This means that the test image should be centered to the mean of the training images, which required me to pass the scaler mean down to this step. The projection residual is then calculated as described and the results are shown below.



```
Subject 1 Test Residuals:
        S1 Eigenfaces: 5679348.443674623
        S2 Eigenfaces: 30460607.78558686
Subject 2 Test Residuals:
        S1 Eigenfaces: 33791504.40346034
        S2 Eigenfaces: 1955858.096001522
```

From these scores, we can clearly see that the residuals are lowest when the test image and eigenfaces are from the same subject. This makes sense, and we can classify the test image by choosing the subject with the lowest residual.

3. (5 points) Comment if your face recognition algorithm works well and discuss how you would like to improve it if possible.

I think the algorithm works well enough - the residuals are clearly different enough to distinguish which test image belongs to which subject. I believe that more training data would be an impactful improvement. Further, I think that the data should be normalized for lighting. Clearly, the shadowed images significantly effect the eigenfaces, so removing that element would help with the pure facial recognition.

5. To subtract or not to subtract, that is the question [Bonus: 5 points].

In PCA, we have to subtract the mean to form the covariance matrix

$$C = \frac{1}{m}\sum_{i=1}^{m}(x^i - \mu)(x^i - \mu)^T$$

before finding the weight vectors, where $\mu = \frac{1}{m}\sum_{i=1}^{m} x^i$. For instance, we let

$$Cw^1 = \lambda_1 w^1$$

where λ_1 is the largest eigenvalue of C, and w^1 is the corresponding largest eigenvector.

Now suppose Prof. X insisting not subtracting the mean, and uses the eigenvectors of

$$\tilde{C} = \frac{1}{m}\sum_{i=1}^{m} x^i {x^i}^T$$

to form the weight vectors. For instance, she lets $\tilde{w}^1$ to be such that

$$\tilde{C}\tilde{w}^1 = \tilde{\lambda}_1 \tilde{w}^1$$

where $\tilde{\lambda}_1$ is the largest eigenvalue of $\tilde{C}$.

Now the question is, are they the same (with and without subtract the mean)? Is w^1 equal or not equal to $\tilde{w}^1$? Use mathematical argument to justify your answer.

5.1

I will not be attempting this due to time constraints.

References

[PP08] K. B. Petersen and M. S. Pedersen. *The Matrix Cookbook*. Version 20081110. Oct. 2008. URL: http://www2.imm.dtu.dk/pubdb/p.php?3274.