3 a) $y_1 = XS = X\lambda I = \lambda X$

since there is only a difference in scale between y_1 and X all variance will be preserved: $\frac{1}{N}\sum_{i=0}^{N}(x_i - \bar{x_i})^2 \times 0.7 = \frac{1}{N}\sum_{i=0}^{N}(\tilde{x_i} - \bar{\tilde{x_i}})^2$

$$\frac{1}{N}\sum_{i=0}^{N}(y_i - \bar{y_i})^2 = \frac{1}{N}\cdot\lambda\sum_{i=0}^{N}(x_i - \bar{x_i})$$

→ PCA on the top-5 components will still yield 70% of the variance

b) $y_2 = XR$, $R \in ^{D\times D}$ and $RR^{-1} = I$

considering that D can be interpreted as a rotation of the axis, PCA will still yeld 70% of the variance in a rotated space

- considering that D can be interpreted as a rotation / reflection own an axis, PCA will still yield 70% of the variance in the new space

c) $y_3 = XP$ with $P = diag(5, -5, 5, -5, ..., -5)$

- considering 5 is just a scaling factor we can ignore it (a))
- considering $P^T = P$ and $\frac{1}{25}PP = I$ we encounter the same situation as b)

→ 70% of the variance is kept.

d) $y_4 = XQ$ with $Q = diag(1, 2, ..., D)$

- $QQ \neq I$ such that this is not a reflection / rotation
- $\Sigma_{y_4} = \frac{1}{N}Q^TX^TXQ$ in which the result isn't clean and we can't tell the change in variance without additional information

→ every axis is scaled differently.

e) $y_5 = X + 1_N\mu^T$ where $\mu \in \mathbb{R}^D$

$$1_N\mu^T = N\left\{\begin{bmatrix} \text{---}\ \mu\ \text{---} \\ \text{---------} \\ \vdots \\ \text{---------} \end{bmatrix}\right.$$

we shift every data point by μ but the first step of PCA is removing the mean, thus nullifying the shift operation and yielding the same 70%.

f) $y_4 = XA$, $rank(A) = 5$

$\Rightarrow$ since the rank of a matrix multiplication is the lowest of the two, $rank(y_4) \leq 5$ (in case $rank(X) < 5 \Rightarrow rank(XA) < 5$)

$\Rightarrow$ there will be no more than 5 eigenvalues in the PCA decomposition, so keeping the top 5 will yield 100% of the variance.

4 a) $X = \begin{pmatrix} 4 & 3 & 2 \\ 2 & 1 & -2 \\ 4 & -1 & 2 \\ -2 & 1 & 2 \end{pmatrix}$, $\bar{x} = \frac{1}{N} X^T \cdot 1_N = \frac{1}{N}(8\ 4\ 4)^T = (2\ 1\ 1)^T$

$\tilde{X} = \begin{pmatrix} 2 & 2 & 1 \\ 0 & 0 & -3 \\ 2 & -2 & 1 \\ -4 & 0 & 1 \end{pmatrix} \rightarrow \Sigma_{\tilde{X}} = \frac{1}{N}\tilde{X}^T\tilde{X} = \frac{1}{4}\begin{pmatrix} 24 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 12 \end{pmatrix} = \begin{pmatrix} 6 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$

- $\det(\Sigma_{\tilde{X}} - \lambda I) = 0 \Leftrightarrow \lambda = 6 \vee \lambda = 2 \vee \lambda = 3$

- $(\Sigma_{\tilde{X}} - 6I)\begin{pmatrix} x \\ y \\ z \end{pmatrix} = 0 \Leftrightarrow \begin{cases} \text{———} \\ -4y = 0 \\ -3z = 0 \end{cases}$ $\rightarrow$ eigenvector: $(1\ 0\ 0)$

- $(\Sigma_{\tilde{X}} - 2I)\begin{pmatrix} x \\ y \\ z \end{pmatrix} = 0 \Leftrightarrow \begin{cases} 4x = 0 \\ \text{———} \\ z = 0 \end{cases}$ $\rightarrow$ eigenvector: $(0\ 1\ 0)$

- $(\Sigma_{\tilde{X}} - 3I)\begin{pmatrix} x \\ y \\ z \end{pmatrix} = 0 \Leftrightarrow \begin{cases} 3x = 0 \\ -y = 0 \end{cases}$ $\rightarrow$ eigenvector: $(0\ 0\ 1)$

- var along $(1\ 0\ 0)$: $\frac{\sum (x_i - \bar{x}_i)^2}{N} = \frac{4+4+16}{4} = 6$
- " " $(0\ 1\ 0)$: 2
- " " $(0\ 0\ 1)$: 3

b) $y = \tilde{X}\, \Gamma_{truncated} = \begin{pmatrix} 2 & 2 & 1 \\ 0 & 0 & -3 \\ 2 & -2 & 1 \\ -4 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 0 & -3 \\ 2 & 1 \\ -4 & 1 \end{pmatrix}$

$\hookrightarrow$ var = 6 , $\rightarrow$ var = 3

$\Rightarrow$ fraction of the variance conserved:

$\frac{6+3}{6+2+3} = 9/11 \simeq 0{,}82$

c) consider a new entry which preserves the PCA analysis.

· It would have to conserve the mean and result in a similar Σ

· consider $x_5 = \bar{x} = (2\ 1\ 1)$

obviously the mean stays the same

· $\Sigma_{\bar{x}}' = \frac{1}{5}\begin{pmatrix} 24 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 12 \end{pmatrix} \longrightarrow$ similar to $\Sigma_{\bar{x}}$ (change of scale)

since the eigenvectors will stay equal in a different scale we will encounter precisely the same principal components when adding $x_5 = \bar{x}$ to the dataset

5) $P = MV$

$\Rightarrow$ Leslie : $\begin{pmatrix} 0 & 3 & 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} 0{,}58 & 0 \\ 0{,}58 & 0 \\ 0{,}58 & 0 \\ 0 & 0{,}71 \\ 0 & 0{,}71 \end{pmatrix} = \begin{pmatrix} 1{,}74 & 2{,}84 \end{pmatrix}$

$\rightarrow$ it seems like Leslie prefers the second concept present in our data so we should recommend movies similar to the Titanic, Casablanca seems like a reasonable choice.

6) · $Xw = y \longrightarrow U\Sigma V^T w = y$

$\Rightarrow (U\Sigma V^T)^{-1}(U\Sigma V^T)w = (U\Sigma V^T)^{-1} y$

pseudo inverse

$\Rightarrow V\Sigma^{-1}U^T U\Sigma V^T w = (U\Sigma V^T)^{-1} y$

$\Rightarrow w^* = V\Sigma^{-1}U^T y$

$w^* = V\Sigma^{-1}U^T y$

$\Rightarrow V \in M^{d\times d}$

$\Rightarrow \Sigma^{-1} \in M^{d\times d}$

$\Rightarrow U^T \in M^{d\times m}$

$\Rightarrow y \in \mathbb{R}^m$

$X = U\Sigma V^T$

$m\times d \quad d\times d \quad d\times d$

(X is full rank $\Rightarrow$ number of singular values is d, assuming $d < m$)

$\Rightarrow$ computing w^* starting from the right:

· $U^T y \longrightarrow O(dm)$

· $\Sigma^{-1}(U^T y) \longrightarrow O(d) + O(dm)$

· $V(\underbrace{\Sigma^{-1}U^T y}_{\in \mathbb{R}^d}) \longrightarrow O(d^2)$

$O(dm) + O(d) + O(dm) + O(d^2) = O(md)$ //

(assuming $d < m$)

res

January 12, 2022

1 Programming task 10: Dimensionality Reduction

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

1.1 Exporting the results to PDF

Once you complete the assignments, export the entire notebook as PDF and attach it to your homework solutions. The best way of doing that is 1. Run all the cells of the notebook. 2. Export/download the notebook as PDF (File -> Download as -> PDF via LaTeX (.pdf)). 3. Concatenate your solutions for other tasks with the output of Step 2. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

Make sure you are using `nbconvert` Version 5.5 or later by running `jupyter nbconvert --version`. Older versions clip lines that exceed page width, which makes your code harder to grade.

1.2 PCA

Given the data in the matrix X your tasks is to: * Calculate the covariance matrix Σ. * Calculate eigenvalues and eigenvectors of Σ. * Plot the original data X and the eigenvectors to a single diagram. What do you observe? Which eigenvector corresponds to the smallest eigenvalue? * Determine the smallest eigenvalue and remove its corresponding eigenvector. The remaining eigenvector is the basis of a new subspace. * Transform all vectors in X in this new subspace by expressing all vectors in X in this new basis.

1.2.1 The given data X

[ ]:
```
X = np.array([(-3,-2),(-2,-1),(-1,0),(0,1),
              (1,2),(2,3),(-2,-2),(-1,-1),
              (0,0),(1,1),(2,2), (-2,-3),
              (-1,-2),(0,-1),(1,0), (2,1),(3,2)])
```

1.2.2 Task 1: Calculate the covariance matrix Σ

[ ]:
```
def get_covariance(X):
    """Calculates the covariance matrix of the input data.

    Parameters
    ----------
    X : array, shape [N, D]
        Data matrix.

    Returns
    -------
    Sigma : array, shape [D, D]
        Covariance matrix

    """
    n = X.shape[0]
    mean = X.T @ np.ones(n) / n
    return X.T @ X / n - np.outer(mean, mean)
```

1.2.3 Task 2: Calculate eigenvalues and eigenvectors of Σ.

[ ]:
```
def get_eigen(S):
    """Calculates the eigenvalues and eigenvectors of the input matrix.

    Parameters
    ----------
    S : array, shape [D, D]
        Square symmetric positive definite matrix.

    Returns
    -------
    L : array, shape [D]
        Eigenvalues of S
    U : array, shape [D, D]
        Eigenvectors of S

    """

    return np.linalg.eig(S)
```

1.2.4 Task 3: Plot the original data X and the eigenvectors to a single diagram.

Note that, in general if u_i is an eigenvector of the matrix M with eigenvalue λ_i then $\alpha \cdot u_i$ is also an eigenvector of M with the same eigenvalue λ_i, where α is an arbitrary scalar (including $\alpha = -1$).

Thus, the signs of the eigenvectors are arbitrary, and you can flip them without changing the meaning of the result. Only their direction matters. The particular result depends on the algorithm used to find them.

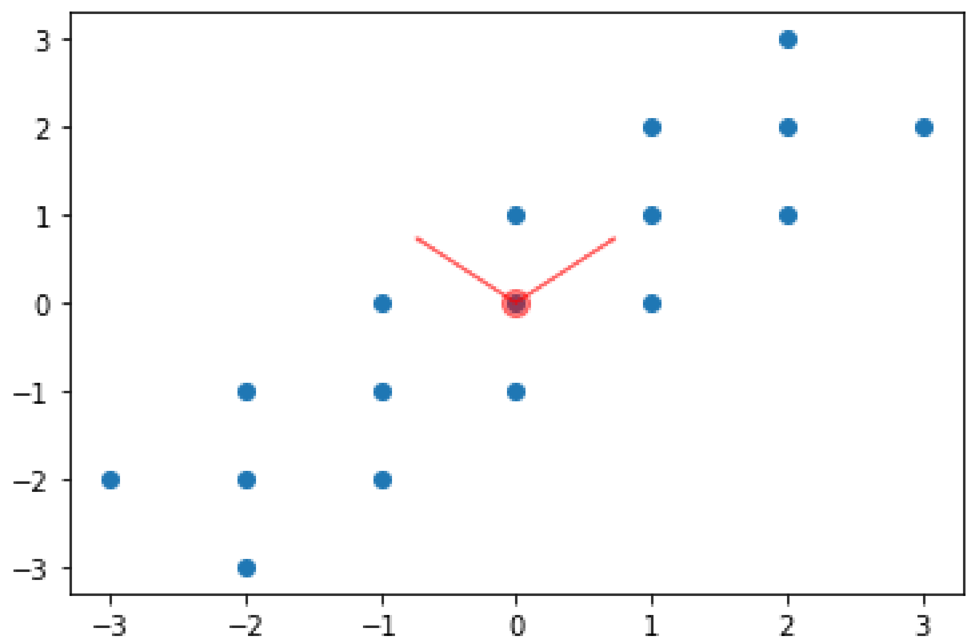
[ ]:
```
# plot the original data
plt.scatter(X[:, 0], X[:, 1])

# plot the mean of the data
mean_d1, mean_d2 = X.mean(0)
plt.plot(mean_d1, mean_d2, 'o', markersize=10, color='red', alpha=0.5)

# calculate the covariance matrix
Sigma = get_covariance(X)
# calculate the eigenvector and eigenvalues of Sigma
L, U = get_eigen(Sigma)

plt.arrow(mean_d1, mean_d2, U[0, 0], U[1, 0], width=0.01, color='red', alpha=0.
 ↪5)
plt.arrow(mean_d1, mean_d2, U[0, 1], U[1, 1], width=0.01, color='red', alpha=0.
 ↪5)
```

[ ]:
```
<matplotlib.patches.FancyArrow at 0x7f01bf864b80>
```



What do you observe in the above plot? Which eigenvector corresponds to the smallest eigenvalue?

Write your answer here:

We can see that the variation along the negatively-sloped eigenvector is the smallest, thus corresponding to the smallest eigenvalue out of the two.

1.2.5 Task 4: Transform the data

Determine the smallest eigenvalue and remove its corresponding eigenvector. The remaining eigenvector is the basis of a new subspace. Transform all vectors in X in this new subspace by expressing all vectors in X in this new basis.

[ ]:
```
def transform(X, U, L):
    """Transforms the data in the new subspace spanned by the eigenvector␣
 ↪corresponding to the largest eigenvalue.

    Parameters
    ----------
    X : array, shape [N, D]
        Data matrix.
    L : array, shape [D]
        Eigenvalues of Sigma_X
    U : array, shape [D, D]
        Eigenvectors of Sigma_X

    Returns
    -------
    X_t : array, shape [N, 1]
        Transformed data

    """

    idx = np.argmin(L)
    U = np.delete(U,idx,1)
    return X @ U
```

[ ]:
```
X_t = transform(X, U, L)
```

1.3 SVD

1.3.1 Task 5: Given the matrix M find its SVD decomposition $M = U \cdot \Sigma \cdot V$ and reduce it to one dimension using the approach described in the lecture.

[ ]:
```
M = np.array([[1, 2], [6, 3],[0, 2]])
```

[ ]:
```
def reduce_to_one_dimension(M):
    """Reduces the input matrix to one dimension using its SVD decomposition.

    Parameters
    ----------
    M : array, shape [N, D]
        Input matrix.

    Returns
```

```
    -------
    M_t: array, shape [N, 1]
        Reduce matrix.

    """
    u, sig, _ = np.linalg.svd(M, full_matrices=False)
    dim = np.argmin(sig)
    u = np.delete(u, dim, 1)
    sig = np.delete(sig, dim)

    return u * sig
```

[ ]:
```
M_t = reduce_to_one_dimension(M)
```