Assignment 6

1. Suppose $\boldsymbol{A}$ is an n-by-n symmetric, *rank-one* matrix with right singular vectors $\boldsymbol{v}_k, k = 1, 2, \ldots, n$. Show that the power method converges to $\boldsymbol{v}_1$ and determine the number of iterations needed to converge within 1% of $\boldsymbol{v}_1$ if your initial vector $\boldsymbol{b}_0 = \frac{1}{\sqrt{n}} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$.

A is $n \times n$, symmetric and rank 1 so

$A = V \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix} V^T = V \Lambda V^T$. σ_1^2 is the only non-zero eigenvalue, meaning v_1 is the only non-zero vector from the singular vectors.

Then from the power iteration we get

$b_k = \frac{A b_{k-1}}{\|A b_{k-1}\|_2}$ which becomes $\frac{A^k b_0}{\|A^k b_0\|_2}$

By expressing b_0 with respect to the eigenvectors V we get $b_0 = Vg$.

Then $b_k = \frac{V \Lambda^k g}{\|V \Lambda^k g\|_2}$ which we can express as

$\frac{\lambda_1^k g_1 \underline{v}_1}{\|\lambda_1^k g_1 \underline{v}_1\|_2}$ because the matrix A is rank 1, so only the first eigenvalue λ_1 is non zero

This will simplify to $\frac{\underline{v}_1}{\|\underline{v}_1\|_2} = \underline{v}_1$, so the power iteration converges to $\underline{v}_1$.

Finally, because the matrix A is rank 1, only 1 iteration is needed to converge to v_1. There is only one singular vector/eigenvector which is v_1. In the step $\frac{V \Lambda^k g}{\|V \Lambda^k g\|_2}$, Λ^k only contains λ_1^k.

Therefore, doing this once will result in $\frac{\lambda_1 g_1 \underline{v}_1}{\|\lambda_1 g_1 \underline{v}_1\|_2}$

which will equal v_1.

2. Use the starter script available for download on the course page. The script loads the 1000 three-dimensional data points in `sdata.csv` into a 1000-by-3 matrix `X`. The second section of the code displays the data using a scatter plot format. We wish to approximate the data using a subspace. Use the rotate tool to view the data cloud from different perspectives.

a) Does the data appear to lie in a low-dimensional subspace? Why or why not? Remember the definition of a subspace.

No, this data is not in a subspace. In the def of subspace, we need to have the origin, point (0,0,0) in the space for it to be a subspace. In the area of the current data, (0,0,0) lies away from the plotted data points.

b) What could you do to the data so that it lies (approximately) in a low-dimensional subspace?

Since the data appears to form around a line in 3 dimensions, the last aspect of a subspace needed is to include origin (0,0,0). Therefore, we need to shift all data points by the same amount so that (0,0,0) is included in the data cloud.

c) The third section of the code removes the mean (average) value of the 1000 data points. Use the rotate tool to inspect the scatterplot of the data with the mean removed. Does the mean-removed data appear to lie in a low-dimensional subspace?

Yes, this data appears to be in a subspace. The data forms around a line in 3 dimensions so most of the points will be able to be represented by a low dimensional subspace. The origin is also within the cloud, so the requirements for a subspace are met.

d) The fourth section of the code finds the SVD of the mean-removed data. If `a` is a unit-norm vector representing the best one-dimensional subspace for the mean-removed data, complete the line of code to define `a` in terms of the SVD matrices. The fifth section displays the one-dimensional subspace with the mean-removed data. Note that the length of the vector representing the subspace is scaled by the root-mean-squared value of the data for display purposes. Use the rotate tool to inspect the relationship between the subspace and the data, and comment on how well a one-dimensional subspace captures the data.

The vector is near the center of the data cloud and goes in the direction of most variability. Therefore most of the data will be captured and can be represented by this single vector, since the most variation in the data is accounted for with this subspace.

e) Let $\boldsymbol{x}_{zi}, i = 1, 2, \ldots, 1000$ be the individual mean-removed data points and $\boldsymbol{a}$ the unit-norm vector representing the best one-dimensional subspace for the data. Thus, $\boldsymbol{x}_{zi} \approx \boldsymbol{a} w_i$. Find w_i in terms of the SVD matrices $\boldsymbol{U}$, $\boldsymbol{S}$, and $\boldsymbol{V}$.

First principal component uses U_1, σ_1, V_1^T

The vector for the first principal component is V_1^T.

$x_{zi} \approx [U_1]_i \sigma_i V_1^T$

V_1^T is the vector so $a = V_1^T$

Thus $w_i = [U_1]_i \sigma_i$ where $[U_1]_i$ is the ith index of the vector U_1.

f) Now write the original data $\boldsymbol{x}_i, i = 1, 2, \ldots, 1000$ as $\boldsymbol{x}_i \approx \boldsymbol{a} w_i + \boldsymbol{b}$. What is $\boldsymbol{b}$?

$x_i \approx a w_i + b_i$

$x_i \approx x_{zi} + b_i$

$b_i \approx x_i - x_{zi}$ b_i is the difference between the original data and the mean removed data approximation.

So b_i should be the mean and a little bit of error due to x_{zi} not being exactly equal to the original mean removed data points

g) Let $\boldsymbol{E}$ be the difference between $\boldsymbol{X}$ and the rank-one approximation. Find a mathematical expression for $||\boldsymbol{E}||_F^2$ in terms of the singular values of the mean-removed data $\boldsymbol{X}_z$.

$X_1 = u_1 \sigma_1 v_1^T \longrightarrow$ from SVD of $X_z = U, \Sigma, V$

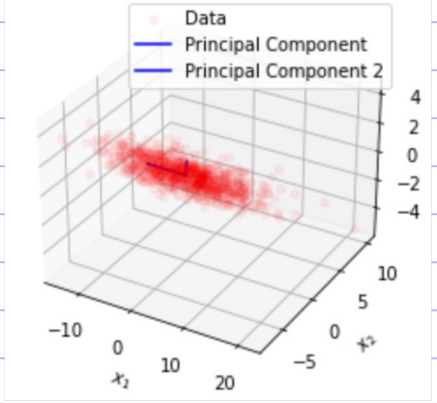
$E = X - X_1$

$= U \Sigma V^T - u_1 \sigma_1 v_1^T$

$= \sum_{i=2}^{3} (u_i \sigma_i v_i^T) = u_2 \sigma_2 v_2^T + u_3 \sigma_3 v_3^T$

$||E||_F^2 = \sum_{i=2}^{3} \sigma_i^2 = \sigma_2^2 + \sigma_3^2$

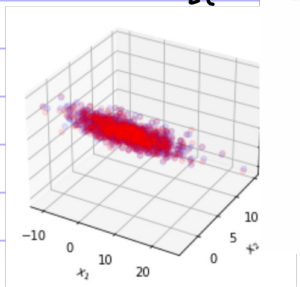
h) Now try a rank-two approximation. Use the SVD to find an orthonormal basis for the best plane containing the mean-removed data. Display the mean-removed data and the bases for the plane.



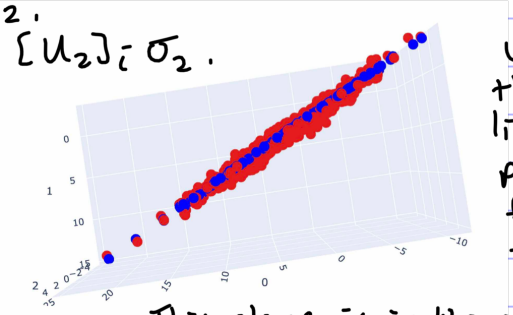
i) Your rank-two approximation for the original data is $\boldsymbol{x}_i \approx \boldsymbol{a}_1 w_{1i} + \boldsymbol{a}_2 w_{2i} + \boldsymbol{b}, i = 1, 2, \ldots, 1000$. Express $w_{2i}, i = 1, 2, \ldots, 1000$ in terms of the SVD of the mean-removed data matrix $\boldsymbol{X}_z$. Display a scatter plot of the original data (in red) and the rank-two approximations in blue. Does the rank-two approximation lie in a plane? Does that plane capture the dominant components of the data?

Second principal component uses u_2, σ_2, V_2^T with $a_2 = V_2^T$.

Thus $w_{2i} = [u_2]_i \sigma_2$.



fixed angle of mat plot lib



when rotated, the blue points all lie in the same plane, as they form a line from the top angle.

This plane is in the middle of the data cloud and varies about the same as the width of the red data, so it captures the dominant components of the data.

j) Let $\boldsymbol{E}$ be the difference between $\boldsymbol{X}$ and the rank-two approximation. Find a mathematical expression for $||\boldsymbol{E}||_F^2$ in terms of the singular values of the mean-removed data $\boldsymbol{X}_z$.

Rank2 approximation using SVD of X_z

$$E = X - \left([u_1 u_2] \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} \right)$$

$$E = X_z - [u_1 \sigma_1 V_1^T + u_2 \sigma_2 V_2^T]$$

$$E = U \Sigma V^T - [u_1 \sigma_1 V_1^T + u_2 \sigma_2 V_2^T] = \sum_{i=3}^{3} u_i \sigma_i V_i^T$$

$$||E||_F^2 = \sum_{i=3}^{3} \sigma_i^2 = \sigma_3^2$$

k) Find and compare the numerical values for $||\boldsymbol{E}||_F^2$ using both the rank-1 and rank-2 approximation.

Rank 1 squared error = 626.69

Rank 2 squared error = 152.946

3. Consider the face emotion classification problem studied previously. Design and compare the performances of the classifiers proposed in **a** and **b**, below. In each case, divide the dataset into 8 equal sized subsets (e.g., examples $1-16$, $17-32$, etc). Use 6 sets of the data to estimate $\boldsymbol{w}$ for each choice of the *regularization parameter*, then select the best value for the regularization parameter by estimating the error on one of the two sets of data held out from training, and finally use the $\boldsymbol{w}$ corresponding to the best value of the regularization parameter to predict the labels of the remaining set of data that was held out. Compute the number of mistakes made on this hold-out set and divide that number by 16 (the size of the set) to estimate the error rate. Note there are $8 \times 7 = 56$ different choices of the two hold-out sets, so repeat this process 56 times and average the error rates across the 56 cases to obtain a final estimate.

a) Truncated SVD. Use the pseudo-inverse $\boldsymbol{V}\boldsymbol{\Sigma}_r^{-1}\boldsymbol{U}^T$, where $\boldsymbol{\Sigma}_r^{-1}$ is computed by inverting the r largest singular values. Hence the regularization parameter r takes values $r = 1, 2, \ldots, 9$.

b) Ridge Regression. Let $\widehat{\boldsymbol{w}}_\lambda = \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$, for the following values of the regularization parameter $\lambda = 0, 2^{-1}, 2^0, 2^1, 2^2, 2^3$, and 2^4. Show that $\widehat{\boldsymbol{w}}_\lambda$ can be computed using the SVD and use this fact in your code.

a) Average error estimate: 11.05%

b) $\hat{w}_\lambda = (X^TX + \lambda I)^{-1} X^T y = V(\Sigma^T\Sigma + \lambda I)^{-1}\Sigma^T U^T y$ using SVD.

The middle section becomes:

$$(\Sigma^T\Sigma + \lambda I)^{-1}\Sigma^T = \begin{bmatrix} \frac{\sigma_1}{\sigma_1^2+\lambda} & & \\ & \ddots & \\ & & \frac{\sigma_n}{\sigma_n^2+\lambda} \end{bmatrix}$$

$$\hat{w}_\lambda = V \begin{bmatrix} \frac{\sigma_1}{\sigma_1^2+\lambda} & & \\ & \ddots & \\ & & \frac{\sigma_n}{\sigma_n^2+\lambda} \end{bmatrix} U^T y$$

which can be computed with SVD in python code.

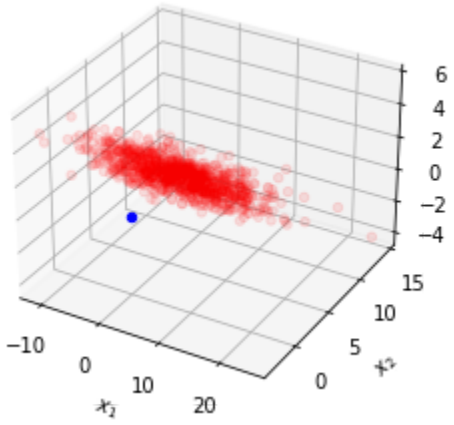
Average error estimate: 4.13%

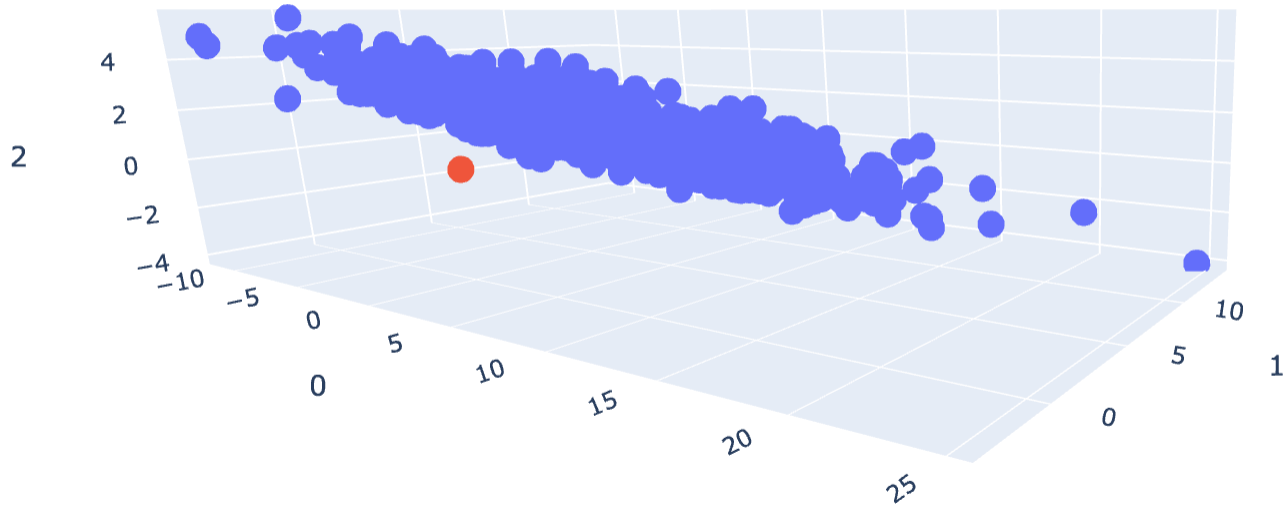
```
# Enable interactive rotation of graph
%matplotlib inline

import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

import plotly.express as px
import plotly.graph_objects as go

# Load data for activity
X = np.loadtxt('sdata.csv',delimiter=',')
rows, cols = np.array(X.shape)

print('Rows of X = ',rows)
print('Cols of X = ',cols)
```

```
Rows of X =  1000
Cols of X =  3
```

2a)

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X[:,0], X[:,1], X[:,2], c='r', marker='o', alpha=0.1)
ax.scatter(0,0,0,c='b', marker='o')
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')

plt.show()

# for rotation in Google colab
fig = px.scatter_3d(X, x=0, y=1, z=2)
fig.add_trace(
    go.Scatter3d(x=[0],
                 y=[0],
                 z=[0],
                 mode='markers')
)
fig.show()
```





2c)

```
# Subtract mean
X_m = X - np.mean(X, 0)


# display zero mean scatter plot
fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_m[:,0], X_m[:,1], X_m[:,2], c='r', marker='o', alpha=0.1)

ax.scatter(0,0,0,c='b', marker='o')
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
```

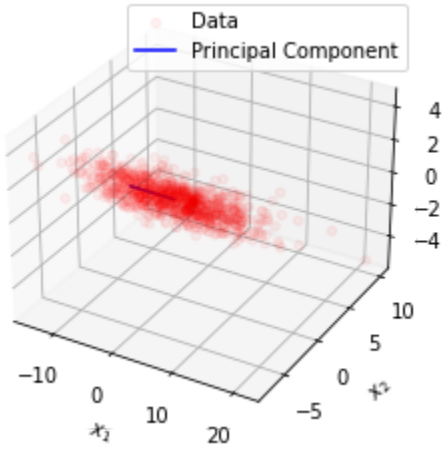
```
ax.set_zlabel('$x_3$')

plt.show()

# for rotation in Google colab
fig = px.scatter_3d(X_m, x=0, y=1, z=2)
fig.add_trace(
    go.Scatter3d(x=[0],
                 y=[0],
                 z=[0],
                 mode='markers')
)
fig.show()
```

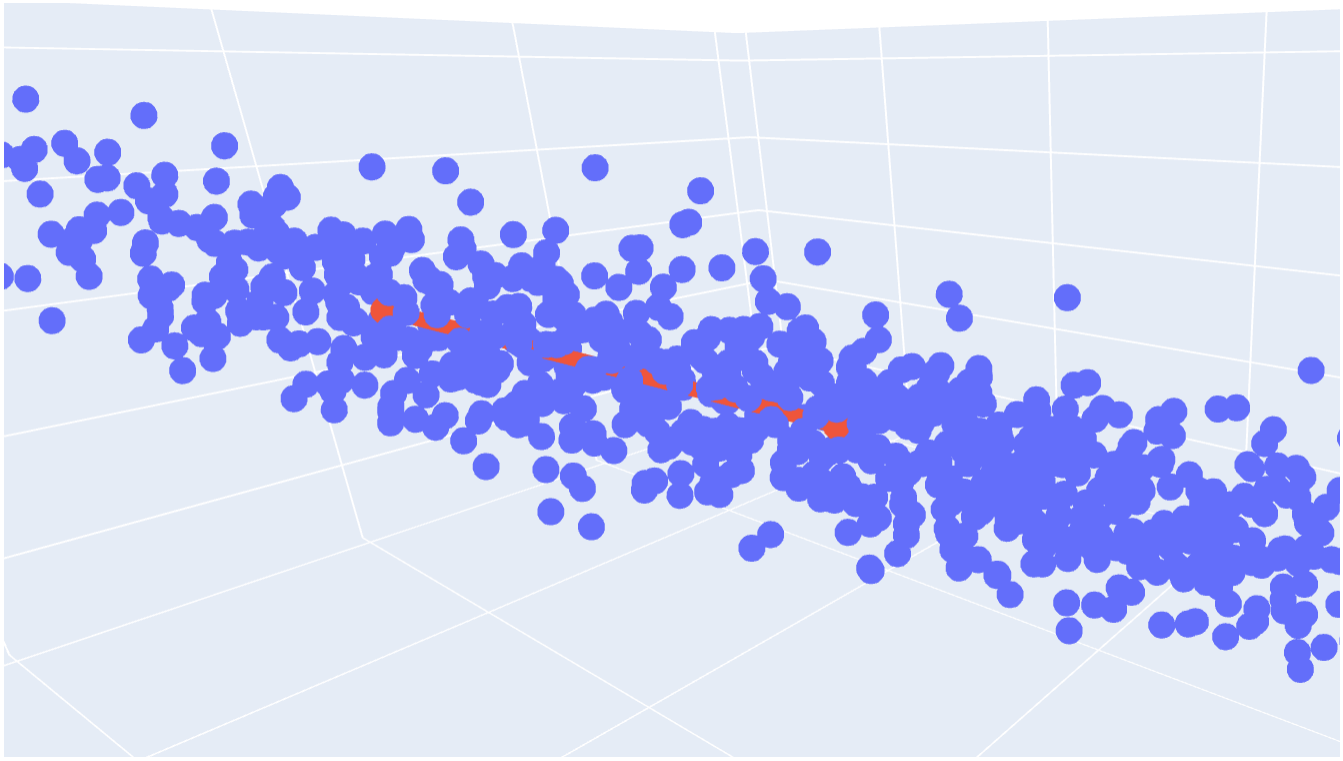
2d)

```
# Use SVD to find first principal component

U,s,VT = np.linalg.svd(X_m,full_matrices=False)

# complete the next line of code to assign the first principal component to a
a = VT[0,:]

print(a)
```

```
     [-0.87325954 -0.43370914  0.2220679 ]
```

```
# display zero mean scatter plot and first principal component

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

#scale length of line by root mean square of data for display
ss = s[0]/np.sqrt(np.shape(X_m)[0])

ax.scatter(X_m[:,0], X_m[:,1], X_m[:,2],
           c='r', marker='o', label='Data', alpha=0.05)

ax.plot([0,ss*a[0]],[0,ss*a[1]],[0,ss*a[2]], c='b',label='Principal Component')

ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')

ax.legend()
plt.show()

# for rotation in Google colab
fig = px.scatter_3d(X_m, x=0, y=1, z=2)
fig.add_trace(
    go.Scatter3d(x=[0,ss*a[0]],
                 y=[0,ss*a[1]],
                 z=[0,ss*a[2]],
                 line=dict(width=15))
)
fig.show()
```





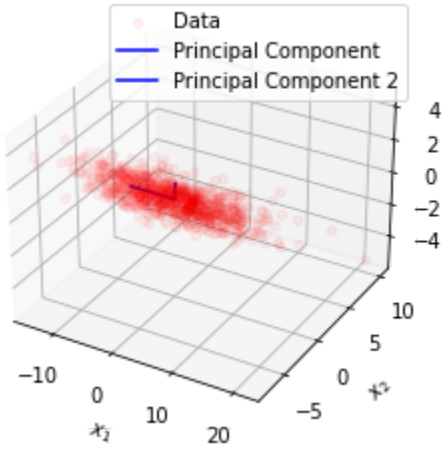
2g) and 2k)

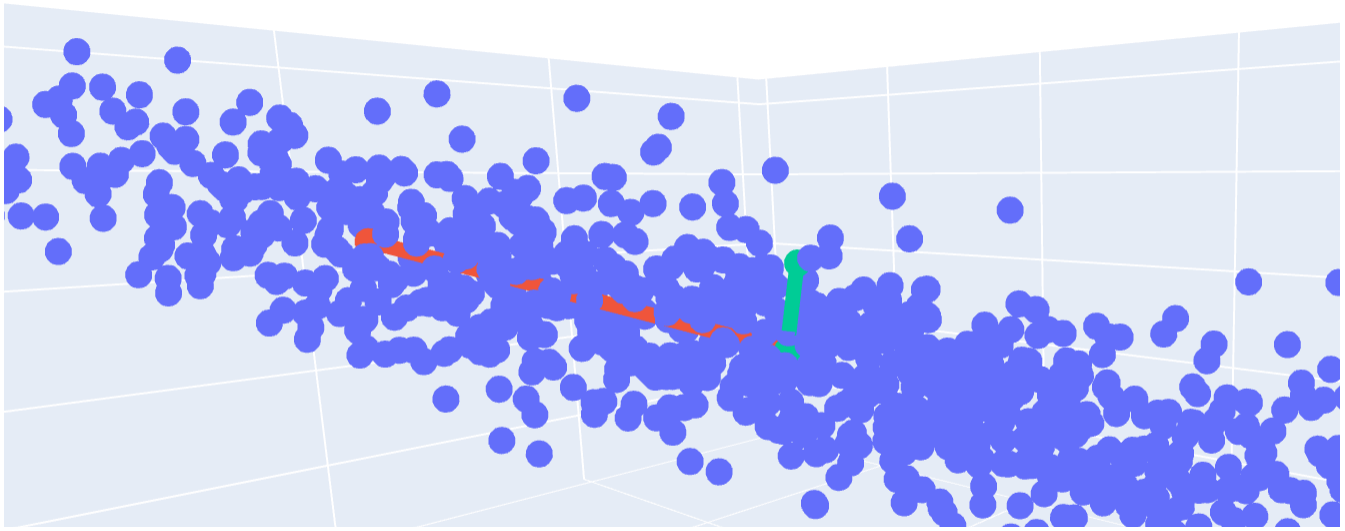
```
U,s,VT = np.linalg.svd(X_m,full_matrices=False)

rank_one = np.array([U[:,0]]).transpose()*s[0]*a
rank_one += np.mean(X, 0)

E = X-rank_one

print(s[1]**2+s[2]**2)
print("Rank 1 squared error: ", np.linalg.norm(E, ord = 'fro')**2)
```

```
626.6899203862782
Rank 1 squared error:  626.6899203862775
```

2h)

```
a2=VT[1,:]

# display zero mean scatter plot and first principal component

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

#scale length of line by root mean square of data for display
ss = s[0]/np.sqrt(np.shape(X_m)[0])

ax.scatter(X_m[:,0], X_m[:,1], X_m[:,2],
           c='r', marker='o', label='Data', alpha=0.05)

ax.plot([0,ss*a[0]],[0,ss*a[1]],[0,ss*a[2]], c='b',label='Principal Component')
ax.plot([0,a2[0]],[0,a2[1]],[0,a2[2]], c='b',label='Principal Component 2')

ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$x_3$')

ax.legend()
plt.show()

# for rotation in Google colab
fig = px.scatter_3d(X_m, x=0, y=1, z=2)
fig.add_trace(
    go.Scatter3d(x=[0,ss*a[0]],
                 y=[0,ss*a[1]],
                 z=[0,ss*a[2]],
                 line=dict(width=15))
)
fig.add_trace(
    go.Scatter3d(x=[0,a2[0]],
                 y=[0,a2[1]],
                 z=[0,a2[2]],
                 line=dict(width=15))
)
fig.show()
```





2i)

```
U,s,VT = np.linalg.svd(X_m,full_matrices=False)

S = np.zeros([2,2])
np.fill_diagonal(S, s[0:2])

rank_two = U[:,0:2]@S@VT[0:2]
rank_two += np.mean(X, 0)

# plots
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X[:,0], X[:,1], X[:,2], c='r', marker='o', alpha=0.1)
ax.scatter(rank_two[:,0], rank_two[:,1], rank_two[:,2],
           c='b', marker='o', alpha=0.1)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
```

```
ax.set_zlabel('$x_3$')

plt.show()

# for rotation in Google colab
fig = px.scatter_3d(X, x=0, y=1, z=2,
                    color_discrete_sequence=px.colors.qualitative.Set1)
fig.add_trace(
    go.Scatter3d(x=rank_two[:,0],
                 y=rank_two[:,1],
                 z=rank_two[:,2],
                 mode='markers',
                 marker=dict(color='blue'))
)
fig.show()
```

2j) and 2k)

```
U,s,VT = np.linalg.svd(X_m,full_matrices=False)

S = np.zeros([2,2])
np.fill_diagonal(S, s[0:2])

rank_two = U[:,0:2]@S@VT[0:2]
rank_two += np.mean(X, 0)

E = X-rank_two

print(s[2]**2)
print("Rank 2 error: ", np.linalg.norm(E, ord = 'fro')**2)
```

```
    152.9455757788646
    Rank 2 error:  152.9455757788646
```

Q3

```
in_data = loadmat("face_emotion_data.mat")
X = in_data['X']
y = in_data['y']

# split into eight sets
eight_sets = np.split(X, 8)
eight_y = np.split(y, 8)
total_error = []
weights_store = []
sum_error = 0

def compute_error(holdout, y_holdout, weights):
  y_hat = np.sign(holdout@weights)
  errors = [0 if i[0]==i[1] else 1 for i in np.hstack((y_holdout, y_hat))]
  return sum(errors)/16

# for the 8 partitions of the data set, remove one
for i in range(8):
  # for the leftover 7 sets, remove another
  # try each of the parameters values on the 6 and test error with the 7th
  for j in range(7):
    # remove 2 sets from the eight
    minus_1 = np.delete(eight_sets, i, axis=0)
```

```
    minus_2 = np.delete(eight_y, i, axis=0)
    training = np.delete(minus_1, j, axis=0).reshape(-1, 9)
    training_y = np.delete(minus_2, j, axis=0).reshape(96, -1)
    # for each value of r
    for r in range(1, 10):
      # weights
      U, s, VT = np.linalg.svd(training.transpose()@training,full_matrices=False)
      S = np.zeros([9,9])
      np.fill_diagonal(S[0:r], 1/s[0:r])
      np.fill_diagonal(S[r:9, r:9], 0)
      pseudo = VT.transpose()@S@U.transpose()
      weights = pseudo@training.transpose()@training_y
      weights_store.append(weights.transpose())
      # error for the 7th set
      total_error.append(compute_error(minus_1[j], minus_2[j], weights))
    # min error for each of the parameters
    minval = min(total_error)
    index = total_error.index(minval)
    # calculate error on last holdout set with best parameter value
    error_best = compute_error(eight_sets[i], eight_y[i],
                               weights_store[index].transpose())
    sum_error += error_best
    total_error=[]

print("Average error: {per}%".format(per=(sum_error/56)*100))
```

```
 Average error: 11.049107142857142%
```

```
X = in_data['X']
y = in_data['y']

# split into eight sets
eight_sets = np.split(X, 8)
eight_y = np.split(y, 8)
total_error = []
weights_store = []
sum_error = 0

def compute_error(holdout, y_holdout, weights):
  y_hat = np.sign(holdout@weights)
  errors = [0 if i[0]==i[1] else 1 for i in np.hstack((y_holdout, y_hat))]
  return sum(errors)/16

lambdas = [0,2**-1,2**0,2**1,2**2,2**3,2**4]

# for the 8 partitions of the data set, remove one
for i in range(8):
  # for the leftover 7 sets, remove another
  # try each of the parameters values on the 6 and test error with the 7th
  for j in range(7):
    # remove 2 sets from the eight
    minus_1 = np.delete(eight_sets, i, axis=0)
    minus_2 = np.delete(eight_y, i, axis=0)
```

```
    training = np.delete(minus_1, j, axis=0).reshape(-1, 9)
    training_y = np.delete(minus_2, j, axis=0).reshape(96, -1)
    # for each lambda
    for r in lambdas:
      # weights
      U, s, VT = np.linalg.svd(training,full_matrices=False)
      S = np.zeros([9,9])
      np.fill_diagonal(S, s/(s**2+r))
      weights = VT.transpose()@S@U.transpose()@training_y
      weights_store.append(weights.transpose())
      # error for the 7th holdout set
      total_error.append(compute_error(minus_1[j], minus_2[j], weights))
    # min error for each of the parameters
    minval = min(total_error)
    index = total_error.index(minval)
    # calculate error on last holdout set with best parameter value
    error_best = compute_error(eight_sets[i], eight_y[i],
                               weights_store[index].transpose())
    sum_error += error_best
    total_error=[]

print("Average error: {per}%".format(per=(sum_error/56)*100))
```

```
 Average error: 4.129464285714286%
```