Assignment 6

1. $b_0 = \frac{1}{\sqrt{n}} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$

$X = \sigma_i U_1 V_1^T = \sigma_i V_1 V_1^T$

$b_1 = \frac{\sigma_i V_1 V_1^T b_0}{\| \sigma V_1 V_1^T b_0 \|_2}$

$= \frac{V_1}{\| V_1 \|_2}$

$= V_1$

$\therefore$ the problem converge at the 1st iteration

2e. $w_i = \sigma_1 U_{1i}$

2g. $\| E \|_F^2 = \sum_{i=1}^{\min(m \times n)} \sigma_i^2$ rank 1 $= \sigma_i^2$

$= \sigma_1^2 + \sigma_2^2 + \sigma_3^2 - \sigma_1^2$

$= \sigma_2^2 + \sigma_3^2$

2j. $\| E \|_F^2 = \sum_{i=1}^{\min(m \times n)} \sigma_i^2$ <u>rank 2</u>

$= \sigma_1^2 + \sigma_2^2 + \sigma_3^2 - (\sigma_1^2 + \sigma_2^2)$

$= \sigma_3^2$

2k. $E_{\text{rank } 1} - E_{\text{rank } 2}$

$\sigma_2^2 + \sigma_3^2 - \sigma_3^2$

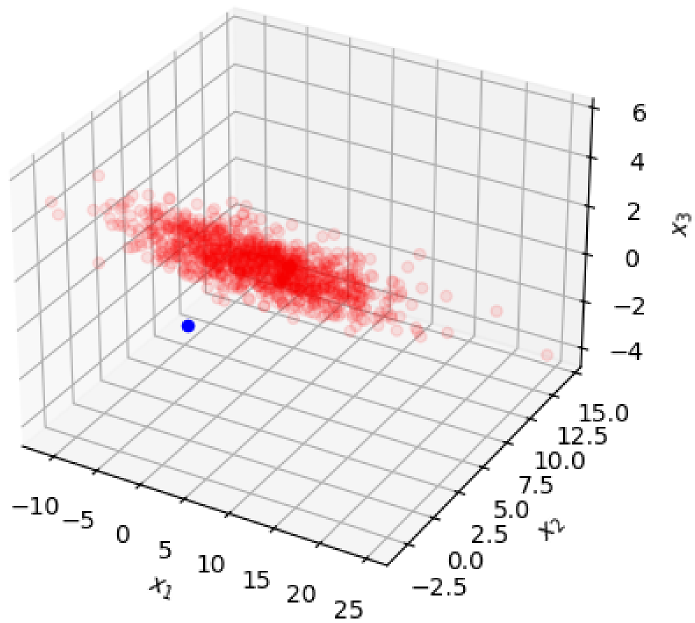
$= \sigma_2^2$

Assign6Starter

March 22, 2024

[130]:
```
# Enable interactive rotation of graph
%matplotlib widget

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

# Load data for activity
X = np.loadtxt('sdata.csv',delimiter=',')
```

[131]:
```
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()
```



1 Question 2a:

No, because the subspace need to cross the origin (0, 0, 0)

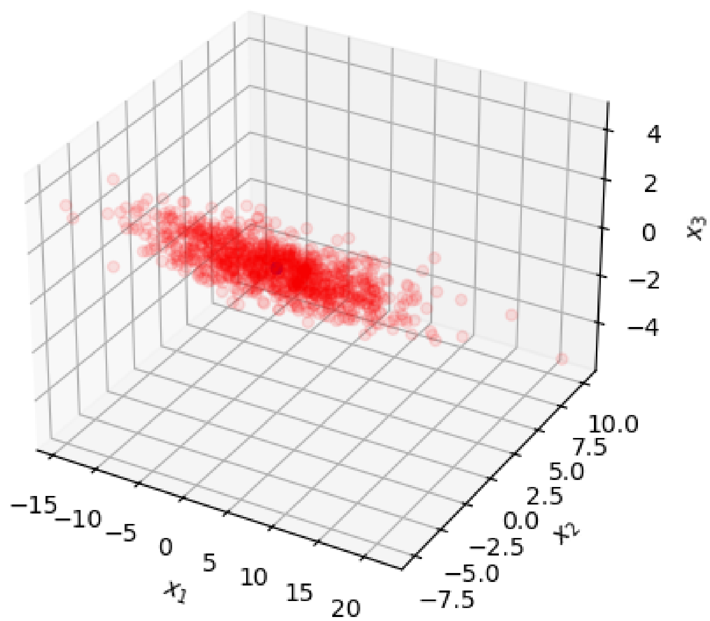
2 Question 2b:

We need to substract the data with mean so the subspace will cross the origin.

[132]:
```
# Subtract mean
X_m = X - np.mean(X, 0)
```

[133]:
```
# 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()
```



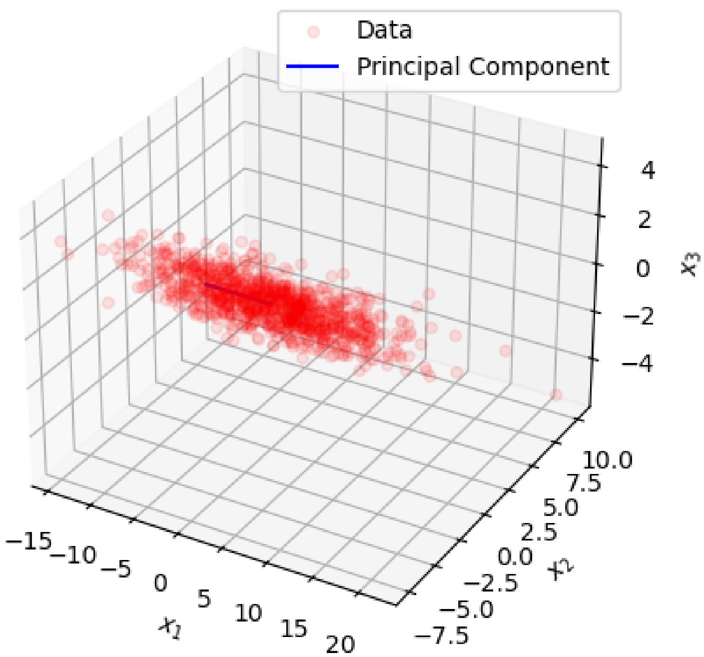
3 Question 2c:

Yes, the mean-removed data appear to lie in a low-dimensional subspace. Because it cross the origin.

[134]:
```
# 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,:]
```

[135]:
```
# 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=0.1)

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()
```



4 Question 2d:

A one-dimensional subspace doesn't display good visualization because there are many overlapping data.

[136]:
```
S = np.diag(s)
S
```

[136]:
```
array([[162.8047015 ,   0.        ,   0.        ],
       [  0.        ,  21.76566894,   0.        ],
       [  0.        ,   0.        ,  12.36711671]])
```

5 Question 2e:

[137]:
```
w = S @ U.T
w
```

[137]:
```
array([[-12.27642044,  -3.80427377,  -2.42719504, …,  -0.50763642,
         -1.50494447,   5.32550802],
       [ -1.89556542,  -0.37776358,   0.10249701, …,  -0.3756265 ,
         -1.13778456,   0.63069539],
       [ -1.06122355,  -0.19912648,  -0.56423101, …,   0.6107223 ,
          0.2801463 ,  -0.10201675]])
```

6 Question 2f:

What is b? b is mean

7 Question 2g: Notes

8 Question 2h:

[138]:
```
# print(VT.shape)
# print(S.shape)
# print(U.shape)
# print(VT[:,:2].shape)
# print(S[:2,:2].shape)
# print(U.T[:2,:].shape)
X = VT.T[:,:2] @ S[:2, :2] @ U.T[:2,:]

X = X.T
print(X.shape)
X
```

```
(1000, 3)
```

[138]:

```
array([[11.18430814,  5.3315948 , -0.88826538],
       [ 3.41454955,  1.65138299, -0.47852885],
       [ 2.09449227,  1.0523074 , -0.63838285],
       …,
       [ 0.53520662,  0.22159313,  0.25147638],
       [ 1.59260018,  0.6570293 ,  0.76899203],
       [-4.80486921, -2.31211678,  0.57110436]])
```

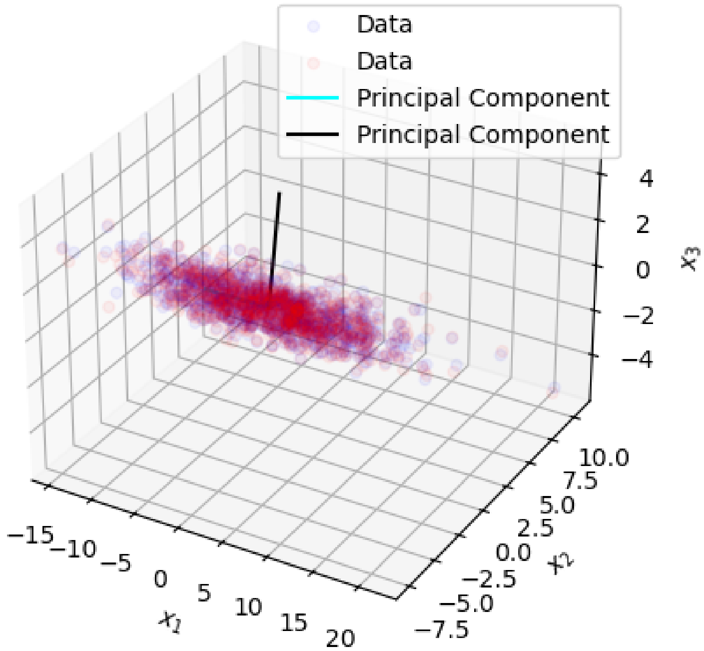
[139]:

```python
# Use SVD to find first principal component
print(X_m.shape)
U,s,VT = np.linalg.svd(X,full_matrices=False)

# complete the next line of code to assign the first principal component to a
a = VT[0,:]
b = VT[1,:]
```

```
(1000, 3)
```

[140]:

```python
# 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)[0])

ax.scatter(X[:,0], X[:,1], X[:,2], c='b', marker='o', label='Data', alpha=0.05)
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='cyan',label='Principal␣
 ↪Component')
ax.plot([0,ss*b[0]],[0,ss*b[1]],[0,ss*b[2]], c='black',label='Principal␣
 ↪Component')

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


ax.legend()
plt.show()
```



9 Question 2i:

The rank 2 approx (blue) is a good approximation. It lies in a plane. The plane capture the dominant components of the data.

10 Question 2k:

Find and compare the numerical values for ||E||2 F using both the rank-1 and rank-2 approximation.

sigma2 square

11 Question 3a:

[141]:
```
import numpy as np
import scipy.io as sio
data = sio.loadmat('face_emotion_data.mat')
X, y = data['X'], data['y']


total_error = []
```

```
for i in range(8):
    for j in range(8):
        if i == j: continue
        test_idx_1 = np.arange(i*16, (i+1)*16)
        test_idx_2 = np.arange(j*16, (j+1)*16)
        train_idx = np.setdiff1d(np.arange(128), test_idx_1)
        train_idx = np.setdiff1d(train_idx, test_idx_2)
        X_train, y_train = X[train_idx, :], y[train_idx, :]
        X_test_1, y_test_1 = X[test_idx_1, :], y[test_idx_1, :]
        X_test_2, y_test_2 = X[test_idx_2, :], y[test_idx_2, :]

        param = None
        lowest = 9223372036854775807

        for r in range(1,10):
            U, s, VT = np.linalg.svd(X_train, full_matrices=False)
            S = np.diag(1/ s[:r])
            approx = VT[:r, :].T @ S @ U[:,:r].T @ y_train

            y_pred = np.sign(X_test_2 @ approx)
            error = []
            for idx, item in enumerate(y_pred):
                if item == y_test_2[idx]:
                    error.append(0)
                else:
                    error.append(1)
            error_percentage = sum(error) / len(error)
            # print(error_percentage)

            if error_percentage < lowest:
                lowest = error_percentage
                param = r

        U, s, VT = np.linalg.svd(X_train, full_matrices=False)
        S = np.diag(1/ s[:param])
        w = VT[:param, :].T @ S @ U[:,:param].T @ y_train

        y_pred_1 = np.sign(X_test_1 @ w)
        error = []
        for idx, item in enumerate(y_pred_1):
            if item == y_test_1[idx]:
                error.append(0)
            else:
                error.append(1)
        error_percentage = sum(error) / len(error)
        # print(error_percentage)
        total_error.append(error_percentage)
```

```python
print("Error SVD: ", sum(total_error)/ 56)
```

```
Error SVD:  0.11160714285714286
```

[142]:
```python
import numpy as np
import scipy.io as sio
data = sio.loadmat('face_emotion_data.mat')
X, y = data['X'], data['y']


total_error = []
for i in range(8):
    for j in range(8):
        param = None
        lowest = 9223372036854775807

        lambda_list = [0, 0.5, 1, 2, 4, 8, 16]

        if i == j: continue
        test_idx_1 = np.arange(i*16, (i+1)*16)
        test_idx_2 = np.arange(j*16, (j+1)*16)
        train_idx = np.setdiff1d(np.arange(128), test_idx_1)
        train_idx = np.setdiff1d(train_idx, test_idx_2)
        X_train, y_train = X[train_idx, :], y[train_idx, :]
        X_test_1, y_test_1 = X[test_idx_1, :], y[test_idx_1, :]
        X_test_2, y_test_2 = X[test_idx_2, :], y[test_idx_2, :]



        for x in lambda_list:
            U, s, VT = np.linalg.svd(X_train, full_matrices=False)
            S = s / (s**2 +x)
            S = np.diag(S)

            w = VT.T @ S @ U.T @y_train

            y_pred = np.sign(X_test_2 @ w)
            error = []
            for idx, item in enumerate(y_pred):
                if item == y_test_2[idx]:
                    error.append(0)
                else:
                    error.append(1)
            error_percentage = sum(error) / len(error)

            if error_percentage < lowest:
```

```
                lowest = error_percentage
                param = x



        U, s, VT = np.linalg.svd(X_train, full_matrices=False)
        S = s / (s**2 + param)
        S = np.diag(S)
        # print(S)
        new_w = VT.T @ S @ U.T @y_train
        y_pred_1 = np.sign(X_test_1 @ new_w)
        error = []
        for idx, item in enumerate(y_pred_1):
            if item == y_test_1[idx]:
                error.append(0)
            else:
                error.append(1)
        error_percentage = sum(error) / len(error)
        total_error.append(error_percentage)

print("Error SVD - Ridge: ", sum(total_error)/ 56)
```

```
Error SVD - Ridge:  0.04799107142857143
```

[ ]: