Asgn 6 ECE 532 Ajar Deep Hayla

1. Since A is rank 1 and symmetric, its singular value decomp is

$$A = \lambda_1 v_1 v_1^T$$

$$b_1 = \frac{Ab_0}{\|Ab_0\|_2} = \frac{\lambda v_1 v_1^T b_0}{\|\lambda_1 v_1 v_1^T b_0\|_2}$$

as $\|\lambda_1 v_1 v_1^T b_0\| = |\lambda v_1^T b_0|$ so

$$b_1 = v_1 \operatorname{sign}\{v_1^T b_0\}$$

Thus in one iteration power method converges to the correct singular vector. The sign doesn't matter, since if v_1 is a singular vec, then $-v_1$ is also a singular vector.

2\. a) Data appears to be concentrated along a line, or even more so, in a plane but since it does not include the origin it's not a subspace.

b) We can recenter the data by removing the mean value from every datapoint. This will center the cloud on the origin & a line/plane approximation will then include the origin.

c) Yes, a line through the origin captures the majority of the variability in data. A plane captures even more.

d) $a = V(:,1)$ or $a = np.transpose(vT)[:,0]$

see plot.

e) X_{xi} is the ith row of matrix X_2.
The rank-1 approx to X_2 is $X_2 \approx U_1 * \sigma_1 * V_1^T$
where U_1 & V_1 are the left & right singular vectors associated with the largest singular value σ_1.
Thus $w_i = [U_1]_i \sigma_1$ where $[U_1]_i$ is the ith entry in U_1.

f) b is the mean that was removed from original data. Note $x_i = x_{xi} + b$

g) We have $X = \sum_{i=1}^{3} \sigma_i u_i v_i^T$ and $X_1 = \sigma_1 u_1 v_1^T$
So $E = \sum_{i=2}^{3} \sigma_i u_i v_i^T$.

Also, $\|E\|_F^2 = \sigma_2^2 + \sigma_3^2$.

h) following prev. steps gives us $\|E\|_F^2 = \sigma_3^2$.

i) We can write $X_2 \approx U_1 * \sigma_1 * V_1^T + U_2 * \sigma_2 * V_2^T$
Thus extracting the ith row of X_2 and rewriting it as a column vector we have

$X_{x_i} = V_1 \times \sigma_1 \times |U_1|_i + V_2 \times \sigma_2 \times |U_2|_i$

where $|U_1|_i$ is the ith entry in U_1 &

$|U_2|_i$ is the ith entry in U_2.

Hence $a_1 = V_1$ & $a_2 = V_2$

we get $w_{1i} = |U_1|_i \sigma_1$ and

$w_{2i} = |U_2|_i \sigma_2$

i) $||E||_F^2 = \sigma_3^2$ from before.

where σ_3 is the smallest singular value of given 3-1000 matrix X_2.

k) Rank-1 squared errors is 626.69.

Rank-2 Absolute errors is 152.95.

Normalizing gives us relative squared errors of 0.023 & 0.006 respectively.

3. We get average error rate of 0.1116 for SVD (truncated).

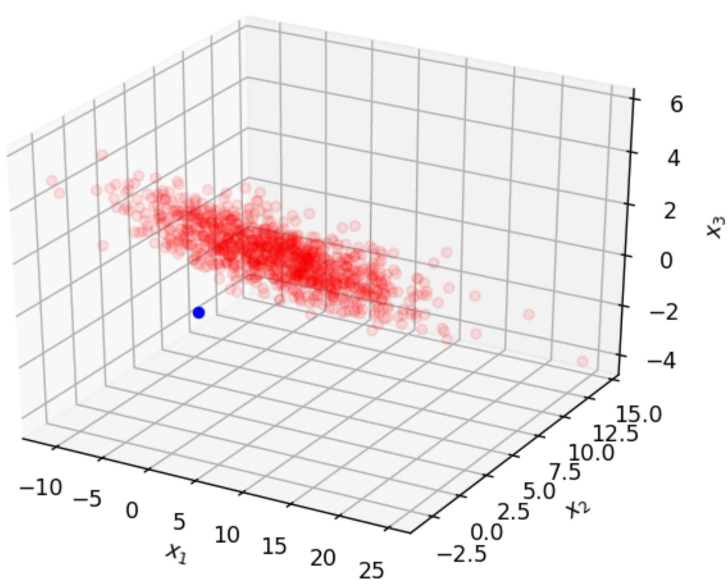
We get average error rate of 0.048 for ridge regression

Thus ridge regression is better here.

In [1]:
```
# Enable interactive rotation of graph
%matplotlib notebook

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=',')
```

In [2]:
```
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()
```

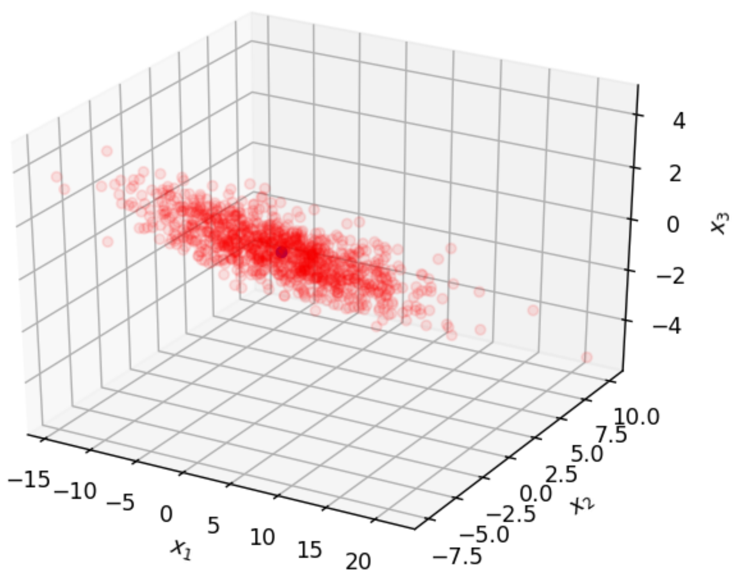
Figure 1



In [3]:
```
# Subtract mean
X_m = X - np.mean(X, 0)
```

In [4]:
```
# 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()
```

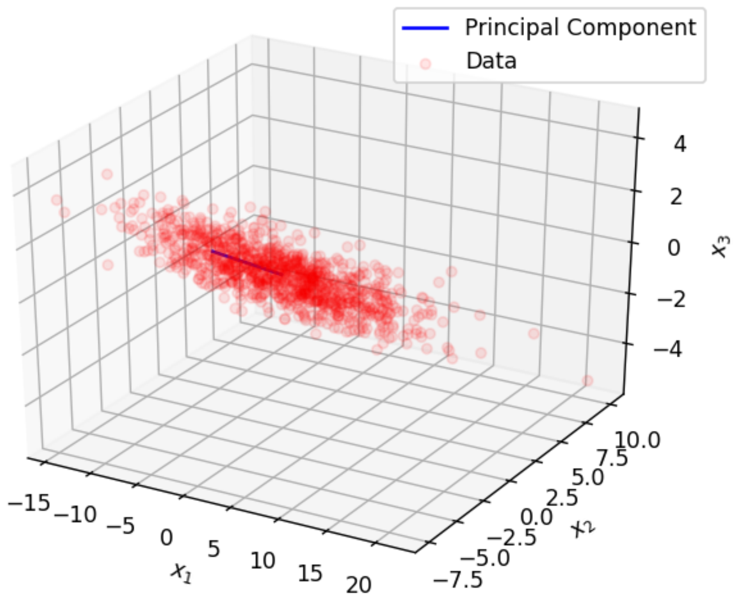
Figure 2



In [5]:
```
# 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 = np.transpose(VT)[:,[0]]
```

In [6]:
```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_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()
```

Figure 3




```
C:\Users\Ayan Deep Hazra\miniconda3\Lib\site-packages\numpy\lib\stride_tricks.py:341: VisibleDeprecationWarning:
Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with d
ifferent lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creatin
g the ndarray.
  array = np.array(array, copy=False, subok=subok)
C:\Users\Ayan Deep Hazra\miniconda3\Lib\site-packages\numpy\core\_asarray.py:171: VisibleDeprecationWarning: Cre
ating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with diff
erent lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating t
he ndarray.
  return array(a, dtype, copy=False, order=order, subok=True)
C:\Users\Ayan Deep Hazra\miniconda3\Lib\site-packages\numpy\core\_asarray.py:102: VisibleDeprecationWarning: Cre
ating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with diff
erent lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating t
he ndarray.
  return array(a, dtype, copy=False, order=order)
```

a

In [7]:
```
import numpy as np

import scipy.io as sio

data = sio.loadmat('face_emotion_data.mat')

X, y = data['X'], data['y']

err_sum = 0

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, :]

        min_err, min_r, min_w = np.inf, -1, None

        for r in range(1,10):

            U, s, VT = np.linalg.svd(X_train)

            w = VT[:r, :].T@np.diag(1/s[:r])@U[:,:r].T@y_train

            err_ = np.mean(np.sign(X_test_1@w) != y_test_1)

            if err_< min_err:

                min_err, min_r, min_w = err_, r, w

        err_sum += np.mean(np.sign(X_test_2@min_w) != y_test_2)

print(err_sum/8/7)
```

```
0.11160714285714286
```

b

In [8]:

```
import numpy as np

import scipy.io as sio

data = sio.loadmat('face_emotion_data.mat')

X, y = data['X'], data['y']

err_sum = 0

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, :]

        min_err, min_r, min_w = np.inf, -1, None

        for la in [0]+[2.**i for i in range(-1,5)]:

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

            w = VT.T@np.diag(s/(s**2+la))@U.T@y_train

            err_ = np.mean(np.sign(X_test_1@w) != y_test_1)

            if err_< min_err:

                min_err, min_r, min_w = err_, r, w

        err_sum += np.mean(np.sign(X_test_2@min_w) != y_test_2)

print(err_sum/8/7)
```

```
0.04799107142857143
```

In [ ]: