Assignment 5

Damion Huppert

Question 1-a

In [2]:
```
import numpy as np

U = (1/2) * np.array([[1,  1], [1, -1], [-1, 1], [1,  1]])
V = (1/np.sqrt(2)) * np.array([[1,  1], [1, -1]])
y = np.array([1, 0, 0, 1])

def compute_results(gamma):
    S = np.array([[1, 0], [0, gamma]])
    X = U @ S @ V.T
    cond_number = 1 / gamma
    w, residuals, rank, s = np.linalg.lstsq(X, y, rcond=None)
    w_norm_squared = np.linalg.norm(w, 2) ** 2

    return cond_number, w_norm_squared

results_1 = compute_results(0.1)
results_2 = compute_results(1e-8)
print(f"Gamma = 0.1: Condition Number: {results_1[0]} ||w||^2_2: {results_1[
print(f"Gamma = 1e-8: Condition Number: {results_2[0]} ||w||^2_2: {results_2
```

```
Gamma = 0.1: Condition Number: 10.0 ||w||^2_2: 101.0
Gamma = 1e-8: Condition Number: 100000000.0 ||w||^2_2: 9999999916955192.0
```

Question 1-b

In [3]:
```
U = (1/2) * np.array([[1,  1], [1, -1], [-1, 1], [1,  1]])
V = (1/np.sqrt(2)) * np.array([[1,  1], [1, -1]])
y = np.array([1, 0, 0, 1])

# Function to compute results for given gamma
def compute_perturbation(gamma, epsilon):
    # Compute X
    S = np.array([[1, 0], [0, gamma]])
    X = U @ S @ V.T
    y_e = np.array([1 + epsilon, 0, 0, 1])
    w_0, _, _, _ = np.linalg.lstsq(X, y, rcond=None)
    w_e, _, _, _ = np.linalg.lstsq(X, y_e, rcond=None)
    w_perturbation = w_e - w_0
    w_perturbation_norm_squared = np.linalg.norm(w_perturbation, 2) ** 2
    return w_0, w_e, w_perturbation_norm_squared

# Compute for epsilon = 0.01 and gamma = 0.1, gamma = 10^-8
perturbation_1 = compute_perturbation(0.1, 0.01)
perturbation_2 = compute_perturbation(1e-8, 0.01)
```

```
print(perturbation_1)
print(perturbation_2)
```

```
(array([ 7.77817459, -6.36396103]), array([ 7.81706547, -6.39578084]), 0.002
5250000000002337)
(array([ 70710678.53215379, -70710677.11794023]), array([ 71064231.92523307,
-71064230.50394846]), 249999998515.75894)
```

When the condition number is lower the norm of w_e is lower

Question 1-c

In [4]:
```
def compute_low_rank_solution(gamma, epsilon):
    # Compute X
    S = np.array([[1, 0], [0, gamma]])  # Sigma matrix
    X = U @ S @ V.T

    U_svd, S_svd, Vt_svd = np.linalg.svd(X, full_matrices=False)

    S_inv_low_rank = np.zeros_like(S_svd)
    S_inv_low_rank[0] = 1 / S_svd[0]

    X_low_rank_inv = (Vt_svd.T @ np.diag(S_inv_low_rank) @ U_svd.T)

    w_0_low_rank = X_low_rank_inv @ y

    y_e = np.array([1 + epsilon, 0, 0, 1])
    w_e_low_rank = X_low_rank_inv @ y_e

    w_perturbation_low_rank = w_e_low_rank - w_0_low_rank

    w_perturbation_norm_squared_low_rank = np.linalg.norm(w_perturbation_low

    return w_0_low_rank, w_e_low_rank, w_perturbation_norm_squared_low_rank

# Compute for epsilon = 0.01 and gamma = 0.1, gamma = 10^-8 using r = 1
low_rank_results_1 = compute_low_rank_solution(0.1, 0.01)
low_rank_results_2 = compute_low_rank_solution(1e-8, 0.01)

print(low_rank_results_1)
print(low_rank_results_2)
```

```
(array([0.70710678, 0.70710678]), array([0.71064232, 0.71064232]), 2.5000000
000000218e-05)
(array([0.70710678, 0.70710678]), array([0.71064232, 0.71064232]), 2.5000000
000000218e-05)
```

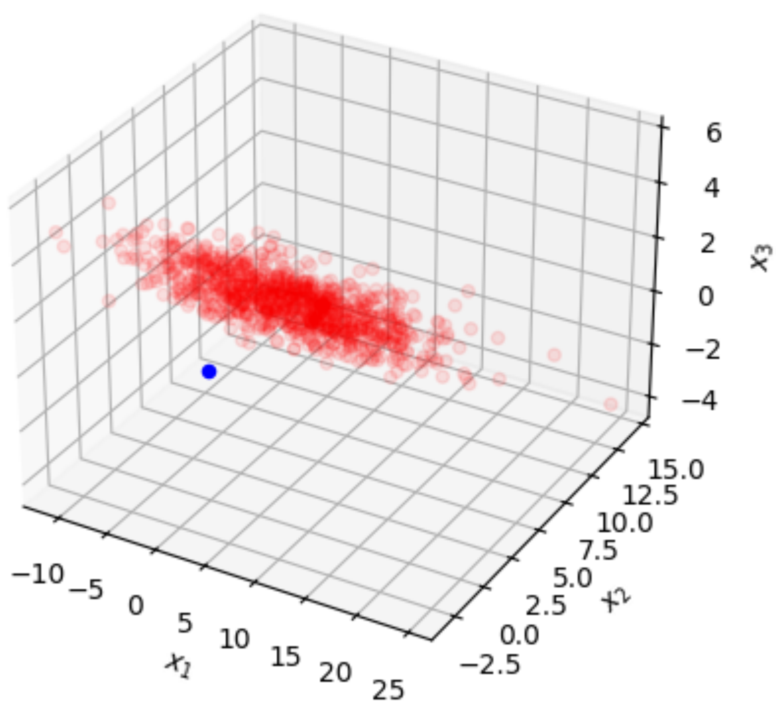
The low rank approx. make the values of w standard for both condition numbers

In [5]:
```
# Enable interactive rotation of graph
%matplotlib widget
```

```python
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 [6]:
```python
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



Question 2-a

It does not lay in LOW dimensional subspace because it is not aligned with the origin. We would need more and 2 subspaces to span the data.

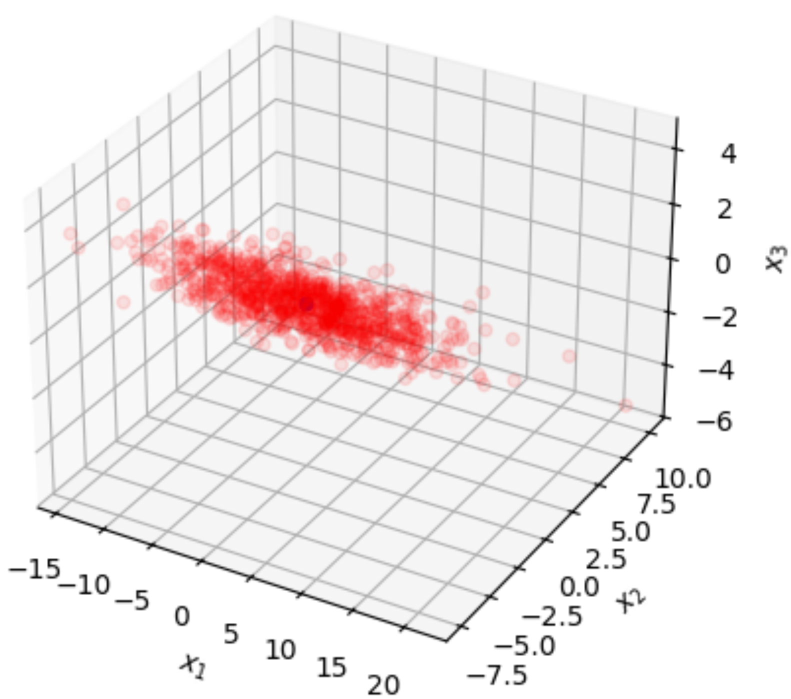
Question 2-b

If we transformed it to have 0 mean it could lie in a low-dimensional subspace

In [7]:
```
# Subtract mean
X_m = X - np.mean(X, 0)
```

In [8]:
```
# 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



Question 2-c

The new data appears to be in a low dimensional subspace

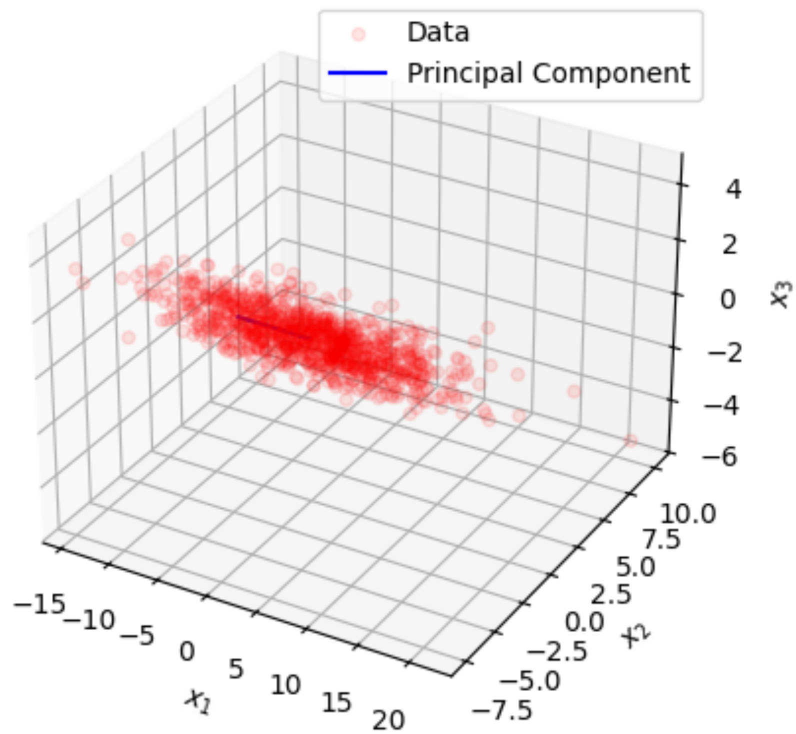
Question 2-d

In [9]:
```
# 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 = VT[0, :]
a
```

Out[9]:
```
array([-0.87325954, -0.43370914,  0.2220679 ])
```

In [10]:
```
# 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', al

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

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


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

Figure



The one dimensional subspace catures the data very well. It seems to point through the middle off all the data

Question 2-e

$$w_i = a^T x_{zi}$$

$$w = U[:, 0]S[0]$$

Question 2-f

b is the mean of X

Question 2-g

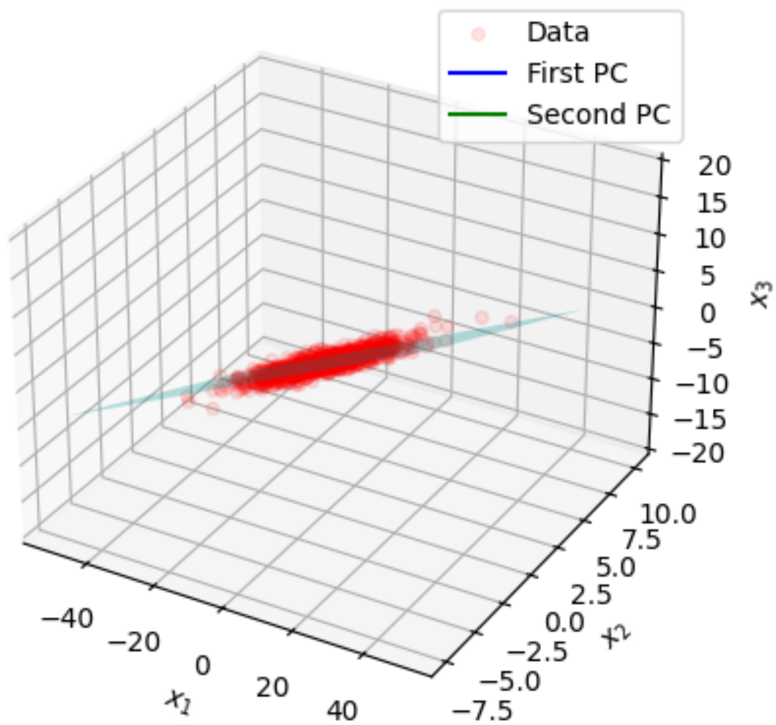
$$||E||_F^2 = \sum_{i=2}^{r}(S_{ii}^2)$$

Question 2-h

In [11]:

```
# Assume X_m is the mean-removed data matrix (1000 x 3)
U, s, VT = np.linalg.svd(X_m, full_matrices=False)

# First two principal components
a1 = VT[0, :]  # First principal component
a2 = VT[1, :]  # Second principal component

# Scale length of line by root mean square of data
ss1 = s[0] / np.sqrt(np.shape(X_m)[0])
ss2 = s[1] / np.sqrt(np.shape(X_m)[0])

# Generate a plane using a1 and a2
grid_x = np.linspace(-ss1*10, ss1*10, 100)
grid_y = np.linspace(-ss2*10, ss2*10, 100)
X_plane, Y_plane = np.meshgrid(grid_x, grid_y)
Z_plane = (a1[2] * X_plane + a2[2] * Y_plane)  # Plane equation using basis

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

# Scatter plot of the mean-removed data
ax.scatter(X_m[:, 0], X_m[:, 1], X_m[:, 2], c='r', marker='o', alpha=0.1, la

# Plot first two principal components
ax.plot([0, ss1 * a1[0]], [0, ss1 * a1[1]], [0, ss1 * a1[2]], c='b', label='
ax.plot([0, ss2 * a2[0]], [0, ss2 * a2[1]], [0, ss2 * a2[2]], c='g', label='

# Plot the plane
ax.plot_surface(X_plane, Y_plane, Z_plane, color='cyan', alpha=0.3)

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

plt.show()
```

Figure

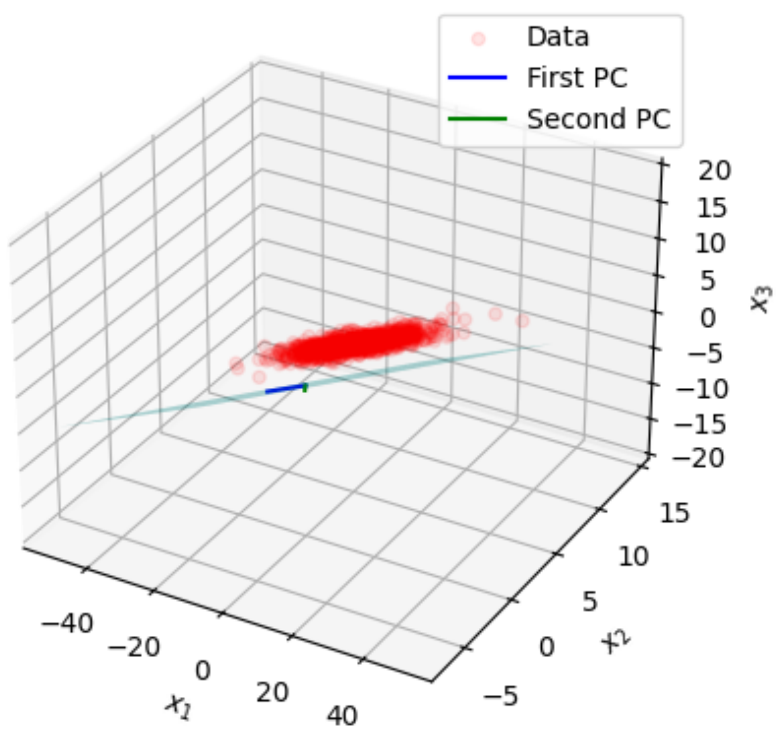


Question 2-i

In [12]:
```
# Assume X_m is the mean-removed data matrix (1000 x 3)
U, s, VT = np.linalg.svd(X_m, full_matrices=False)

# First two principal components
a1 = VT[0, :]  # First principal component
a2 = VT[1, :]  # Second principal component

# Scale length of line by root mean square of data
ss1 = s[0] / np.sqrt(np.shape(X_m)[0])
ss2 = s[1] / np.sqrt(np.shape(X_m)[0])

# Generate a plane using a1 and a2
grid_x = np.linspace(-ss1*10, ss1*10, 100)
grid_y = np.linspace(-ss2*10, ss2*10, 100)
X_plane, Y_plane = np.meshgrid(grid_x, grid_y)
Z_plane = (a1[2] * X_plane + a2[2] * Y_plane)  # Plane equation using basis

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

# Scatter plot of the OG data
```

```
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c='r', marker='o', alpha=0.1, label='D

# Plot first two principal components
ax.plot([0, ss1 * a1[0]], [0, ss1 * a1[1]], [0, ss1 * a1[2]], c='b', label='
ax.plot([0, ss2 * a2[0]], [0, ss2 * a2[1]], [0, ss2 * a2[2]], c='g', label='

# Plot the plane
ax.plot_surface(X_plane, Y_plane, Z_plane, color='cyan', alpha=0.3)

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

plt.show()
```

Figure



The 2 rank approx does lie in a plane.

That plane does captue the dominant components of the data.

Question 2-j

$$||E||_F^2 = \sum_{i=3}^{r}(S_{ii}^2)$$

Question 2-k

In [13]:
```
U, s, VT = np.linalg.svd(X_m, full_matrices=False)
E1_F_squared = np.sum(s[1:] ** 2)
E2_F_squared = np.sum(s[2:] ** 2)

print(f"||E||_F^2 for Rank-1 Approximation: {E1_F_squared:.4f}")
print(f"||E||_F^2 for Rank-2 Approximation: {E2_F_squared:.4f}")
```

```
||E||_F^2 for Rank-1 Approximation: 626.6899
||E||_F^2 for Rank-2 Approximation: 152.9456
```

$||E||_F^2$ for Rank-2 Approximation is smaller