**Compressing Rap vs Classical Music**

**May 14, 2018**

**Jimmy Lemkemeier**

**Overview**

For our project, Myles Caldwell and I investigated if there was a difference between rap and classical music after compressing into R2 and R3. Our two different compression techniques were Principal Component Analysis (PCA) and neural nets. Both of these techniques take a large vector of data which comprises an audio clip and compresses it into two or three data points. While the PCA can more accurately compress the songs into R2 or R3, we tried using neural nets as well in hopes that there would be interesting differences between the techniques.

**Background**

PCA:

Principle Component Analysis is the most precise way to compress data. It uses the fact that every matrix can be diagonalized using its SVD. This means that every matrix A can be written as a product of three matrices:

A = U∆VT

Here the columns of U and V can be represented by and respectively. Both U and V are orthonormal matrices. The matrix ∆ is a diagonal matrix and the diagonal values can be represented as . These diagonal values are singular values of matrix A and are ordered in descending order starting at .

This SVD equation can be rewritten as:

Since and columns in orthonormal matrices, that means that they are both vectors in the unit circle (or whatever shape a circle is in their dimension). Therefore acts as the scaling factor and shows how much A transforms a unit vector. Since they are ordered in descending order, represents the maximum distance A will transform a unit vector.

The other useful thing about this is the fact that is a basis for the codomain of A. Since all of the columns of A lie in the range of A, every column of A can be written as a combination of . Therefore, we can write each column as follows:

As previously discussed, , or in this case , is paired with a corresponding in decreasing order. So, the coefficient for is the most important, followed by , and so on and so forth. Therefore, , even if it is a long vector, can be graphed in R2 or R3 by taking the first two or three coefficients from the previous equation. These are the values which hold the most information about the vector, and therefore this technique would compress it as accurately as possible.

These coefficients , (which can be written as the matrix Y whose columns represent the coefficients for the corresponding column of A), can be found by solving the following equation:

This equation can be rearranged into:

Since U is an orthonormal matrix, (meaning ), this equation can be simplified further:

So, in order to compress the columns of A into two data points while losing minimum information, solve for Y in this above equation and then take the first two entries in each column of Y. These two entries are the best possible compression of each corresponding column of A.

This works for real world data by combining all of your data into a matrix where each column of the matrix represents one data point. Then, by solving for the Y matrix and examining the first few coefficients, it is possible to compress those data points into less dimensions with minimal information loss.

Norms:

A norm of a vector is a function which evaluates the ‘size’ of a vector. The norms this paper will discuss (first, second, and infinity norm) are from the equation:

In this equation, n is derived from the name of the norm (1-norm, 2-norm…) and m is the size of the vector.

This paper also uses the Frobenius Norm of a vector which is:

In this equation vH refers to the conjugate transpose of the vector.

Fournier coefficients:

Fournier coefficients can be best described as follows. Assume that V is an inner product space with an inner product and that B = is an orthonormal basis for V. Assume then

Skipping over some of the finer details, the Fournier coefficients = when you find by using this equation:

For the purpose of this assignment, when we use Fournier coefficientswe will be using the basis:

We will calculate them by using MATLAB function fft().

Neural Nets:

A neural net is a function which takes an input in terms of a vector, passes it through some equations, and returns an output vector. A neural net is created (or trained) by giving it a list of inputs with corresponding desired outputs. The neural net runs the inputs, tests the error between the output and the desired output, and adjusts its equations accordingly.

Generally, a neural net’s equation can be best visualized as a net of neurons. The input vector is the first row of neurons. Then there are rows of neurons of various lengths in the middle and the output is a row of neurons. Each neuron connects to every neuron in the next row. (See Figure 1).

The connection between each neuron is through the equation w\*x+b where x is the prior neuron’s value and w (known as the weight) and b (known as the bias) are variables adjusted by the neural net. So, each neuron has input in the form of w\*x+b from each neuron in the prior layer. That neuron adds all of that input together and runs it through an activation function.

That activation function can be many things, but we only used a Rectified Linear Unit (ReLU) or a Sigmoid. A ReLU means if that combined input is negative the result is zero. If the combined output is positive, then the result is simply the combined output. (See Figure 2). A Sigmoid activation function means that if the input is below a target level, the output is 0. If the output is above a target level, the output is 1. And if the output is right around the target level the output is between 0 and 1. (See Figure 3). The result of a neuron’s activation function is the output of that neuron which it passes to the next layer of neurons. This whole process creates a neural net.

For our purpose of compressing the songs into two points, we used an autoencoder. An autoencoder is a type of neural net where the desired output is simply the input. The twist is that the middle layer of our neural net will only be two neurons in length. Therefore, the neural net must try to compress an audio clip as accurately as possible into two values and then decompress it to get as close to the original audio clip as possible. This compressing is done by trial and error where the neural net tries to adjust its function to gain as much accuracy as possible. Therefore, neural nets are different (and generally less accurate) than the PCA.

**Data**

For our data, we picked 100 songs: 50 rap songs and 50 classical songs. We picked rap and classical music because we believed that those were very opposite genres. If a compression could tell the difference between any two genres, we hypothesized that it would be able to tell the difference between rap and classical music.

These songs came from a variety of different artists and sub-genres of rap and classical music. We used rap songs that were previously downloaded on Myles Caldwell’s iTunes account, as those were an easily accessible option. The classical songs we individually downloaded from YouTube. Since the songs from iTunes were mp4s and the YouTube songs were mp3s, we converted all of them to mp3s to keep them consistent. The mp3 files were originally stored as stereo-files, meaning for every frame of music there were two values. This meant that the songs were stored in two vectors. We turned them into mono-files, meaning one data point per frame, by taking the average of the two values per frame. This made each song into one vector.

We then lowered the frame rate of each song from 16000 to 8000 which cut down the length of each array. This reduced the amount of data per song while still keeping the music's audio clear. We also cut the songs into .25-second-long clips in order to continue to decrease the amount of data from each song. That length seemed to be the shortest possible length where we were still able to tell the song’s genre. We took the .25-second-long clips at the halfway point in the song. We figured that the middle of the song is most likely to sound stereotypical to that genre of music. For instance, rap songs might have an introduction that sounds similar to classical music, but the middle is almost always clearly rap. After these adjustments, each song was represented by a .25-second-long clip stored as a mono-file with a frame rate of 8000 frames per second. This meant that each song was represented by a vector of length 2000.

**Results**

PCA:

We tried to compress the songs first using PCA. Since PCA is the optimal way to compress the songs, it seemed more likely to work than using neural nets.

In order to do the PCA, we created a matrix (using the songs as columns) that was 2000x100. We then found the SVD of that matrix using MATLAB. Next, we calculated the matrix Y as previously shown in the background portion of this report. By taking the first either two or three values from each Y column, we graphed the compressed versions of the songs.

After graphing PCA for the original data, we found that the rap songs were clustered around the graph’s origin and the classical songs were dispersed wildly throughout the graph. (See Figure 4). Although this at first looked promising as it showed differences between the types of music, we concluded that this was due to a difference in volume. Since we had procured the data from different sources, (iTunes vs YouTube), the default volumes of songs were different.

The volume of an audio clip is determined by the size of the values in the audio clip’s vector. Therefore, in order to recalibrate each song, we divided each value in each song’s vector by a norm of that vector since a norm is a measurement of the size of that vector. That would lower the volume, (or size of the values), proportionally to how loud the song was originally.

We tried numerous different norms to recalibrate the songs, but the first one we tried was the 1-norm. Once we recalibrated it with the 1-norm, we took the PCA of the resulting vectors and graphed the resulting points. (See Figure 5). These graphs were different from the original graph where rap music was clustered around the origin. However, now both genres were equally dispersed throughout the graph with no clear differences between them. This meant we had fixed the volume issue but that the PCA had failed to see a difference between the rap and the classical music.

Next, we tried recalibrating the songs with the 2-norm. We took the PCA of the resulting vectors and graphed the points. (See Figure 6). These graphs lacked any clear clustering as well.

At this point, we began to speculate that the audio clips were too short in length for the PCA to distinguish between the genres. Although it was possible for a human to distinguish between the genres with only .25 seconds of a song, we hypothesized that maybe if we gave the computer a longer clip it would be more successful. We increased the clips’ length from .25 seconds to 2 seconds in order to test that hypothesis.

First, we recalibrated the 2-second-long clip with the 1-norm and took the PCA. We graphed the resulting points. (See Figure 7). Despite increasing the length of the clips, there was still no clear clustering.

Next, we did the same thing but recalibrated the 2-second long clips with the 2-norm. (See Figure 8). This graph still did not distinguish between the types of music.

We then tried recalibrating the 2-second-long clips with the infinity norm. We took the PCA and graphed the resulting points. (See Figure 9). Ever since we started accounting for volume, this was the first graph where there were some differences between the genres. Rap music occupies one plane which runs through the origin and classical music occupies a perpendicular plane which also runs through the origin.

At the time, since this graph looked very similar to the original uncalibrated graph, we speculated that this difference was still due to the initial volume differences between the genres of music. Since the infinity norm is just the largest value of each song vector, we thought using this norm might not remove the volume bias. If song x has values in the range of (1,2) and a louder song y has values in the range of (9,10), dividing by the infinity norm would change song x’s range to (.5, 1) whereas song y’s range would change to (.9, 1). Using this logic, we dismissed these results and continued trying other approaches.

However, upon writing this report, it has occurred to us that this problem would exist when recalibrating the vectors with any norm. A larger volume must mean a larger range in order for any of our recalibrations to truly neutralize the volume issue. However, upon researching the issue, a larger volume does mean a larger range. Therefore, the success of the infinity norm should not be dismissed.

Next, we tried recalibrating the 2-second-long clips by using the Frobenius norm. We took the PCA and graphed the points. (See Figure 10). This had no distinguishable clustering between the genres.

Finally, we took the Frobenius coefficients for each 2-second-long song clip. We took the PCA of the results and graphed the points. (See Figure 11). This graph showed no clustering between the genres.

Neural net:

We initially started working with the neural nets using data which was recalibrated by the 2-norm. This was because we had dismissed the infinity norm’s success and took the 2-norm because we had assumed nothing had worked. We used the .25-second-long clips to keep the amount of data low since more data makes neural nets more difficult.

We trained the neural nets by passing the 100 songs through them a given amount of times (denoted by the variable epochs). The neural net would pass them through in batch sizes of 25 songs, and the error was calculated using the mean squared error. This error was based on the distance between the audio input and the audio output by the neural net (since the desired output was the same as the input). After each batch, the neural network would try to minimize the error by adjusting the weights and biases. The amount the variables were altered each time was determined by a set learning rate. We adjusted the epochs and the learning rate to try to minimize the error at the end of the training. Once the training was finished, we would pass the songs through the neural net one last time and record the two values in the middle layer of the neural net for each song. We graphed those values on a graph in R2 to see how the neural net compressed all the songs and hoped to see clustering between the rap songs and classical songs.

The first neural net we built used a Rectified Linear Unit as its activation function, an optimizer called Gradient Descent Optimizer, and one hidden layer with two nodes.

The error generally started very close to 1 since the initial weights and biases were randomly set close to 0. Since every input vector had a length 1 (as it had been calibrated by the 2-norm) and the initial outputs were close to 0, it made sense that the error (the distance between them) started around 1.

At first, our neural net struggled to learn and would often get stuck early. We assumed that we should add more layers of neurons. So, we added a layer of 50 neurons on both sides of the middle layer. Our new configuration was the input layer of 2000, a layer of 50 neurons, a layer of 2 neurons, a layer of 50 neurons, and an output layer of 2000 neurons.

The next issue we encountered was that the Gradient Descent Optimizer would often

get stuck at a local minimum of .995. In order to get past that local minimum, we had to raise the learning rate much higher. At this high learning rate, the error would get past .995 and continue to fall but it would then sporadically jump to above 1. We solved this issue by switching our optimizer to the Adam Optimizer.

The Adam Optimizer worked better and we fine-tuned the learning rate to about .00005 for optimal learning. At any learning rate higher than that, the error would decrease more rapidly but sporadically jump to above 1; any lower and it would learn more slowly. At 100,000 epochs we were able to get the error down to .954. (See Table 1). We then graphed the middle two neurons for each song. (See Figure 12). However, there was still no clustering between the rap and classical music.

We then changed the activation function for the neurons to a sigmoid function as described in the background portion of this paper. This activation function could handle a much high learning rate. We fine-tuned that learning rate to .0065 using the same reasoning as before. With 100,000 epochs we reduced this error to .950. (See Table 1). We graphed the values of the middle two neurons for each song at the end. (See Figure 13). The corresponding graph still lacked any distinct clustering.

Initially, we had stopped here, thinking that the effort was fruitless given that the PCA had not found any clustering. However, after realizing that the success with infinity norm using the PCA should not have been explained away, we tried to run the neural nets when recalibrating the .25-second-long vectors using the infinity norm. We ran this neural net still using the Adam Optimizer. We experimented with many different activation functions, hidden layer configurations, and learning rates in an effort to lower the error. The lowest error we achieved was while using a ReLU activation function with [100, 2, 100] hidden layers and a learning rate of .00001. (See Table 1). We then graphed the corresponding values. (See Figure 14). This unfortunately did not produce the same clustering as it did with the PCA.

**Conclusion**

In light of our experiments, we are able to conclude that it is possible to differentiate between classical songs and rap songs when compressing the songs into R2 and R3 using PCA. By using the infinity norm, there appears to be loose clustering between the two genres. This is most likely due to the fact that in the same range, a rap song is more likely to have a higher maximum value than a classical song. This fact has an increased impact on the song when you divide the song by the infinity norm (the max value of the song). One explanation for the clustering is that the compression was picking up on this difference. This explanation would make sense because the values of the rap songs should be smaller as a whole, since they were divided by larger numbers, and therefore they are more clustered around the origin. Some of the classical songs are near the origin as well, however some of the other classical songs are more dispersed along the plane of classical music. Presumably, those dispersed songs did not hit as high in their allowed range and therefore their average value was larger. The two clusters overlap with each other heavily thus making predictions of an unknown song fairly difficult. The existence of clustering is an exciting sign nonetheless.

The neural net graph with the infinity norm did not produce the same clustering as the PCA norm. One explanation for this was that the neural net was compressing it into two values whereas the PCA was compressing it to three values. However, every 2D angle on the 3D PCA graph shows distinct clustering. Therefore, neural nets still could have clustered the data in 2D. Another possible explanation is that the neural net was only given .25-second clips and the PCA was given 2-second clips. This seems like a more reasonable explanation given that it is more likely for rap music to hit the maximum value in its given range during a 2-second long clip rather than a .25-second long clip. A third very reasonable explanation is that I am simply not adept enough yet at training neural nets for it to find its optimal solution.

**Future Work**

There are several more experiments which would be interesting to run in light of these results.

First, running the neural net with the 2-second-long clips recalibrated by the infinity norm might allow for the same clustering result that was found with the PCA. Someone could also run PCA with .25-second-long clip recalibrated for the infinity norm and see if it also struggles to find clustering with those short clips.

Secondly, running the PCA and neural net with even longer sample clips (maybe even whole songs) would be an interesting experiment. This would allow more time for the rap songs to hit higher in their ranges, thus creating more of a difference between the songs.

Third, someone could run PCA and neural nets on rap songs with louder beats to see if this exacerbates the effect. If this worked, it would strengthen our explanation.

Fourth, someone could run data analytics on the classical songs which are clustered at the origin in the existing infinity norm graph vs the classical ones which are more dispersed. More specifically looking into their infinity norms and whether the ones closer to the origin hit higher maximum values in their clips.

Fifth, experimenting with devising a norm which would exacerbate these clusters could be intriguing. Maybe a norm similar to the infinity norm, where you sum the top ten largest values in the vector from ten different areas of the vector. Theoretically, a classical song might hit a high value a few times in a short period, but a rap song would be more likely to hit them frequently throughout the song. If the infinity norm in rap songs is the beat, this could be particularly successful. By creating a basis which would more likely be different between the two genres, we could recalibrate our data in a way that would allow the computer to better differentiate between rap and classical music.

Figure 1 (Neural Net Diagram):

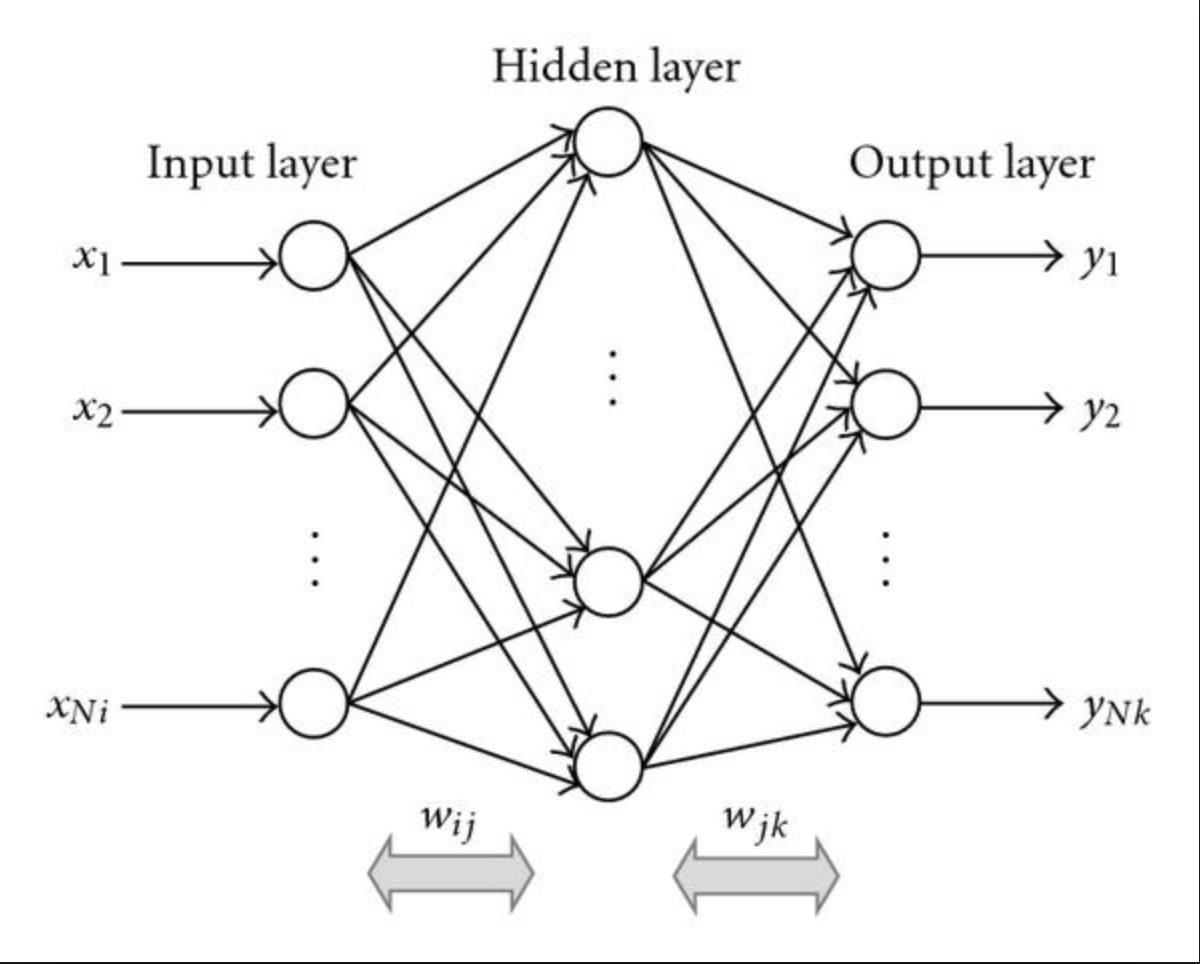


Figure 2 (ReLU Function):

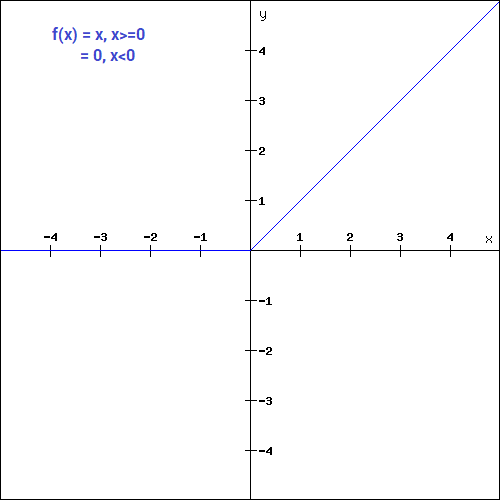


Figure 3 (Sigmoid Function):



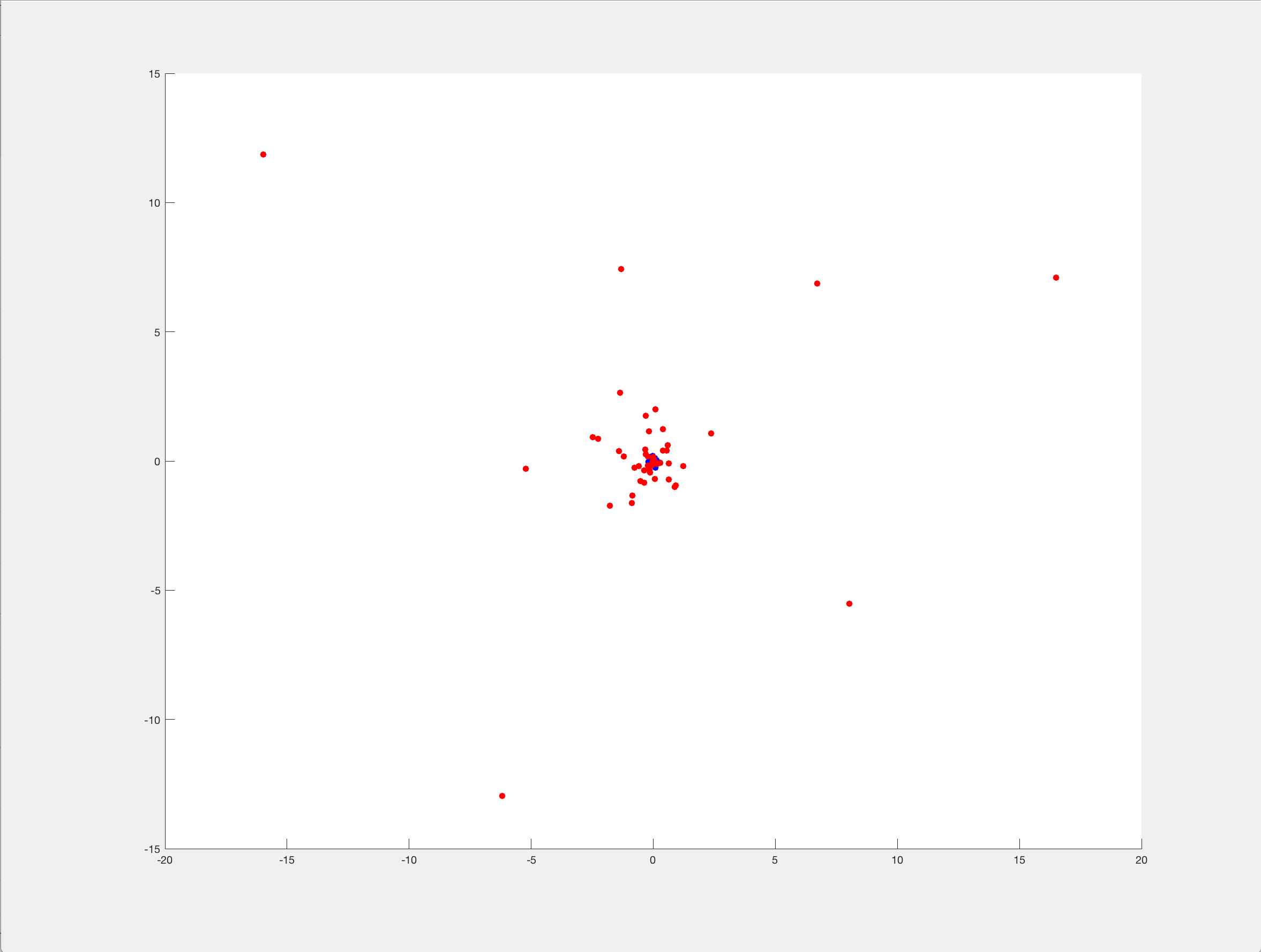
Figure 4a (.25-second uncalibrated 2D PCA):

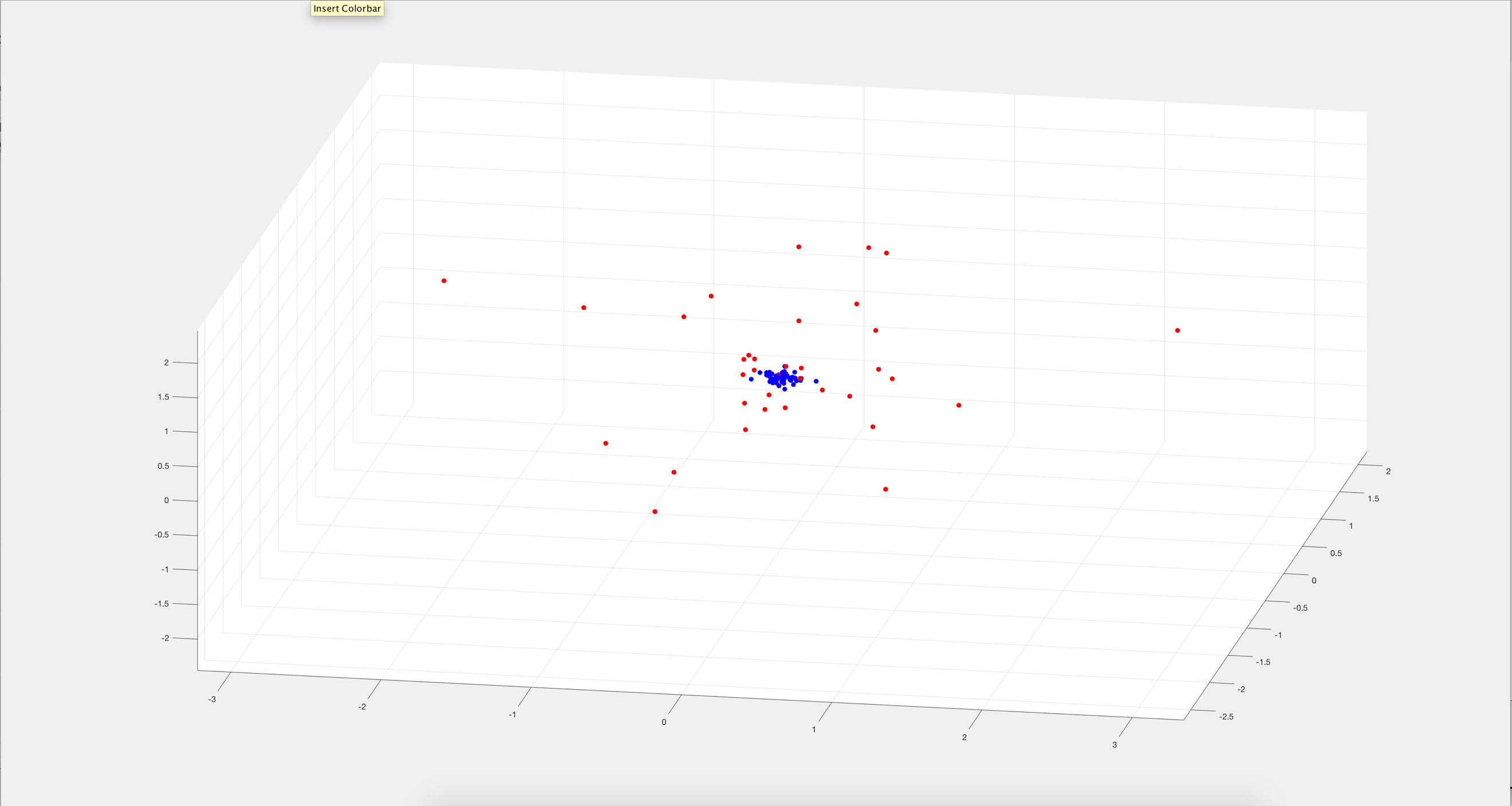
Figure 4b (.25-seconds uncalibrated 3D PCA):

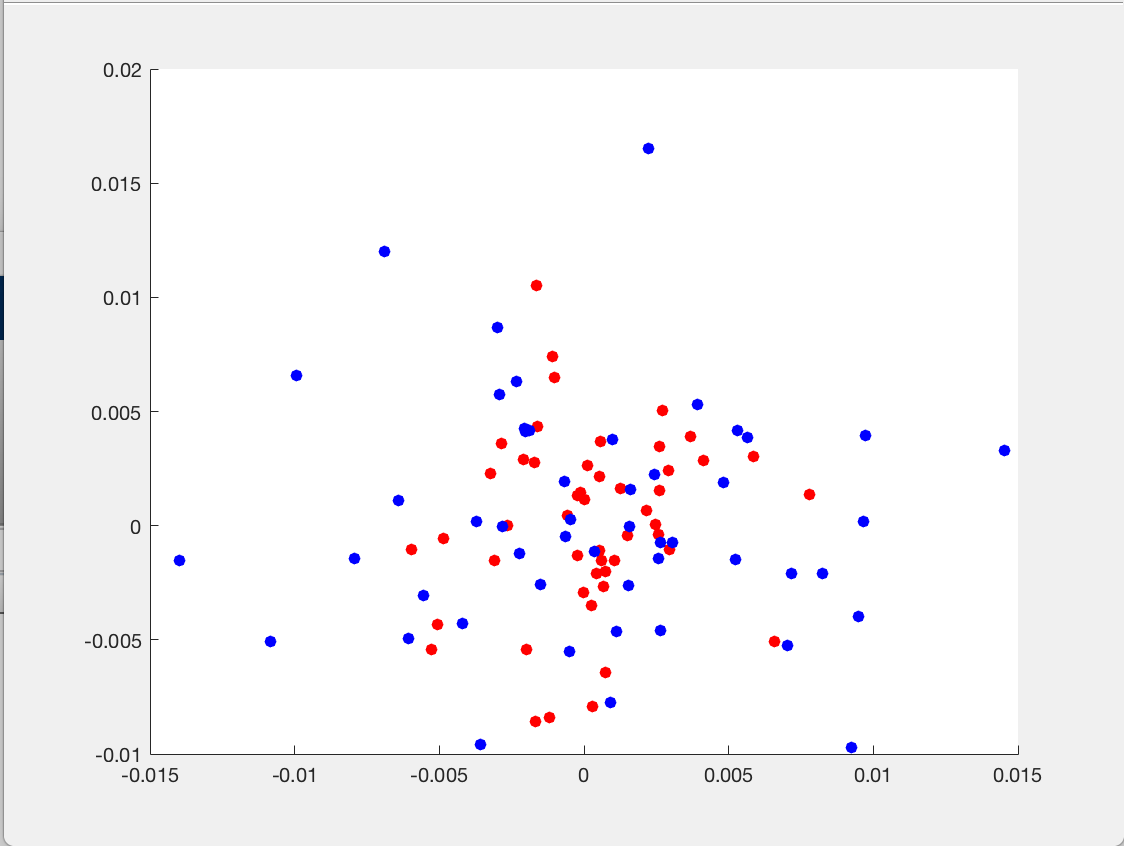
Figure 5a (.25-seconds 1 Norm 2D PCA):

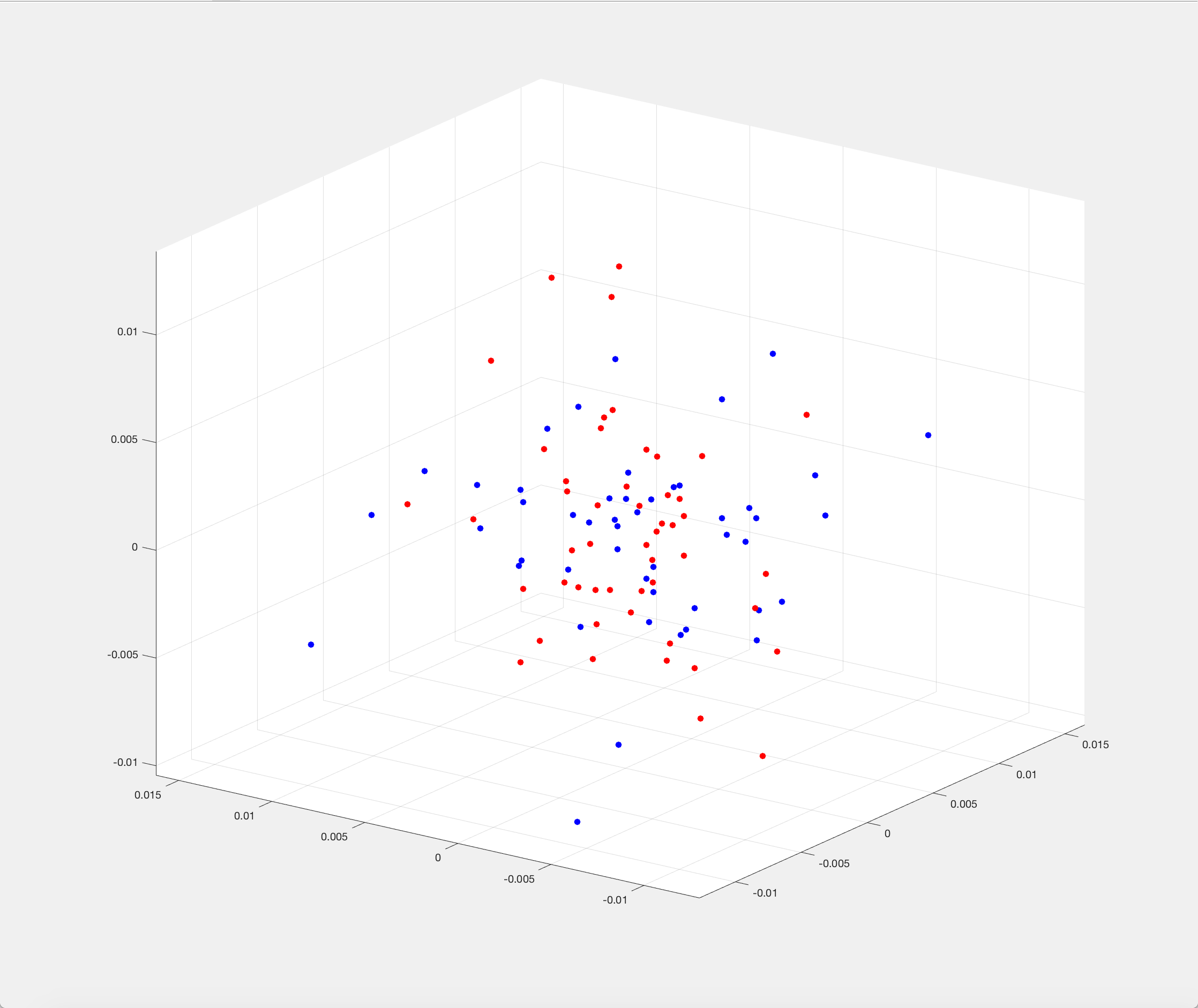
Figure 5b (.25 seconds 1 Norm 3D PCA):

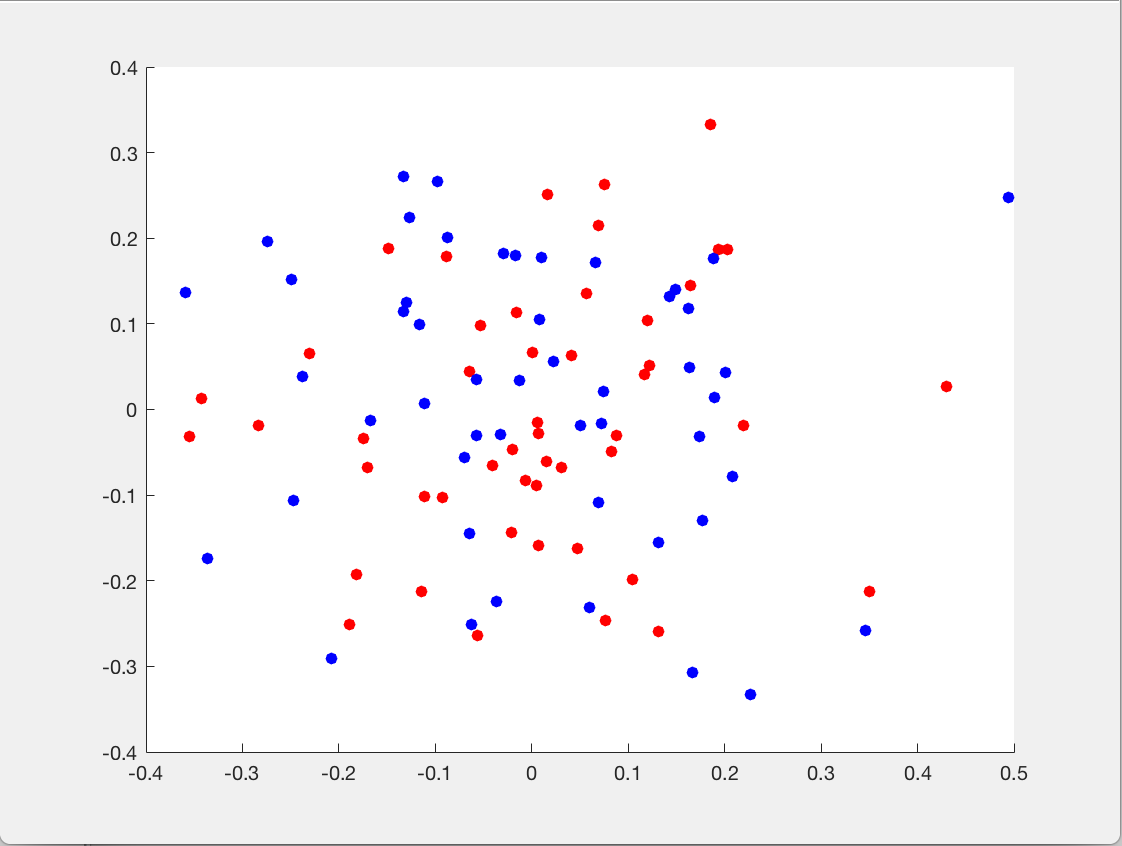
Figure 6a (.25 seconds 2 Norm 2D PCA):

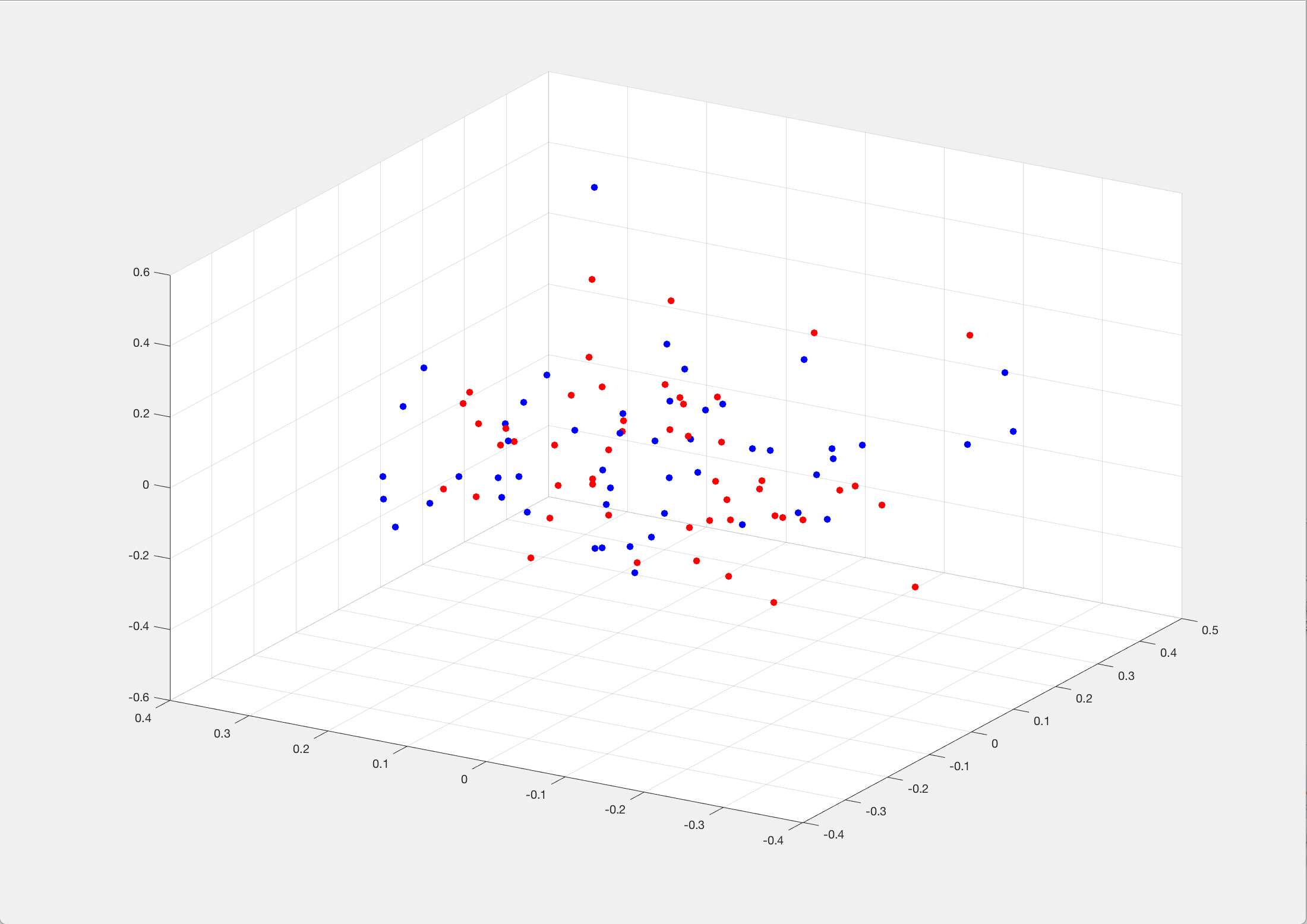
Figure 6b (.25 seconds 2 Norm 3D PCA):

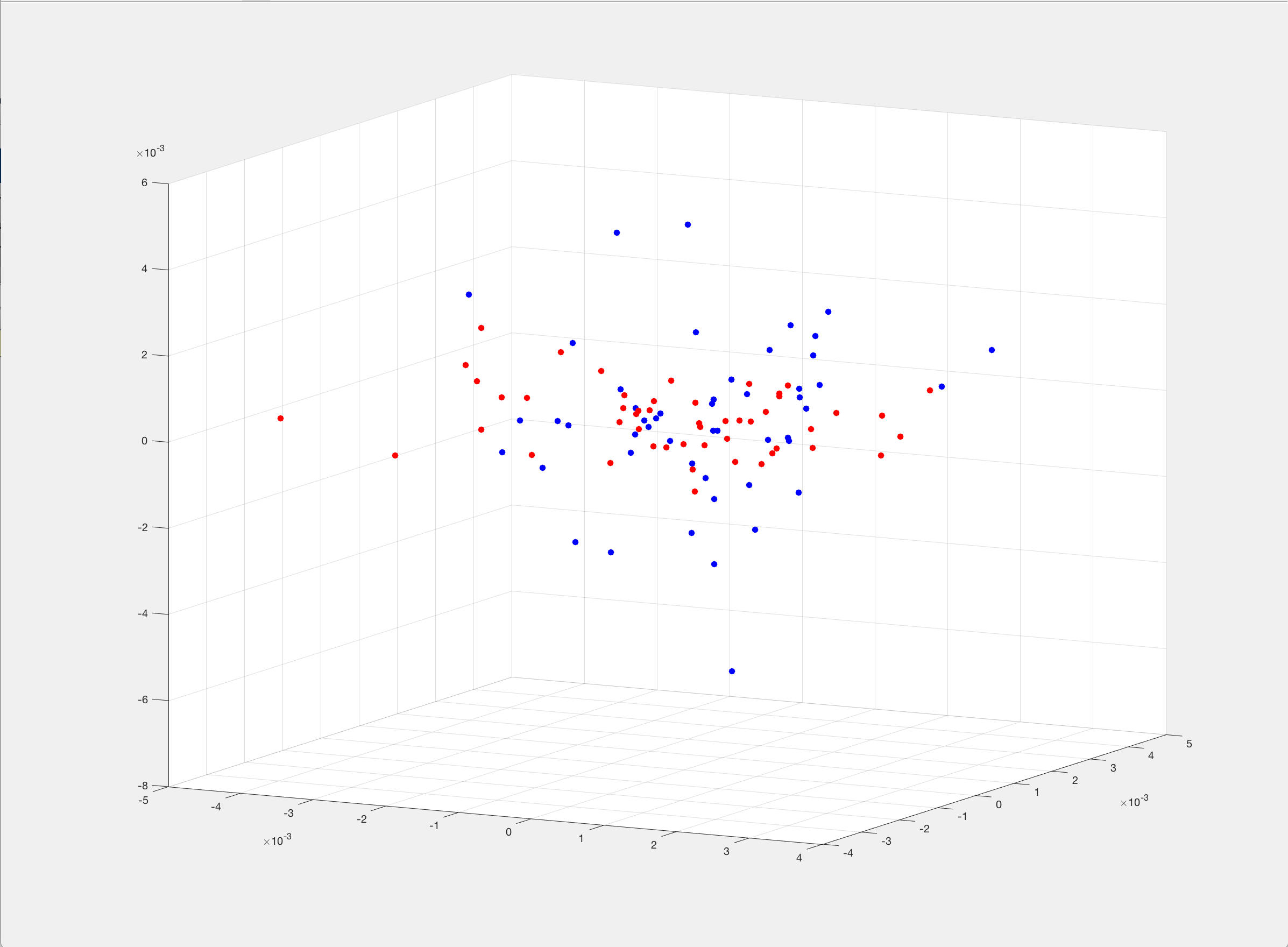
Figure 7 (2 Seconds 1 Norm 3D PCA):

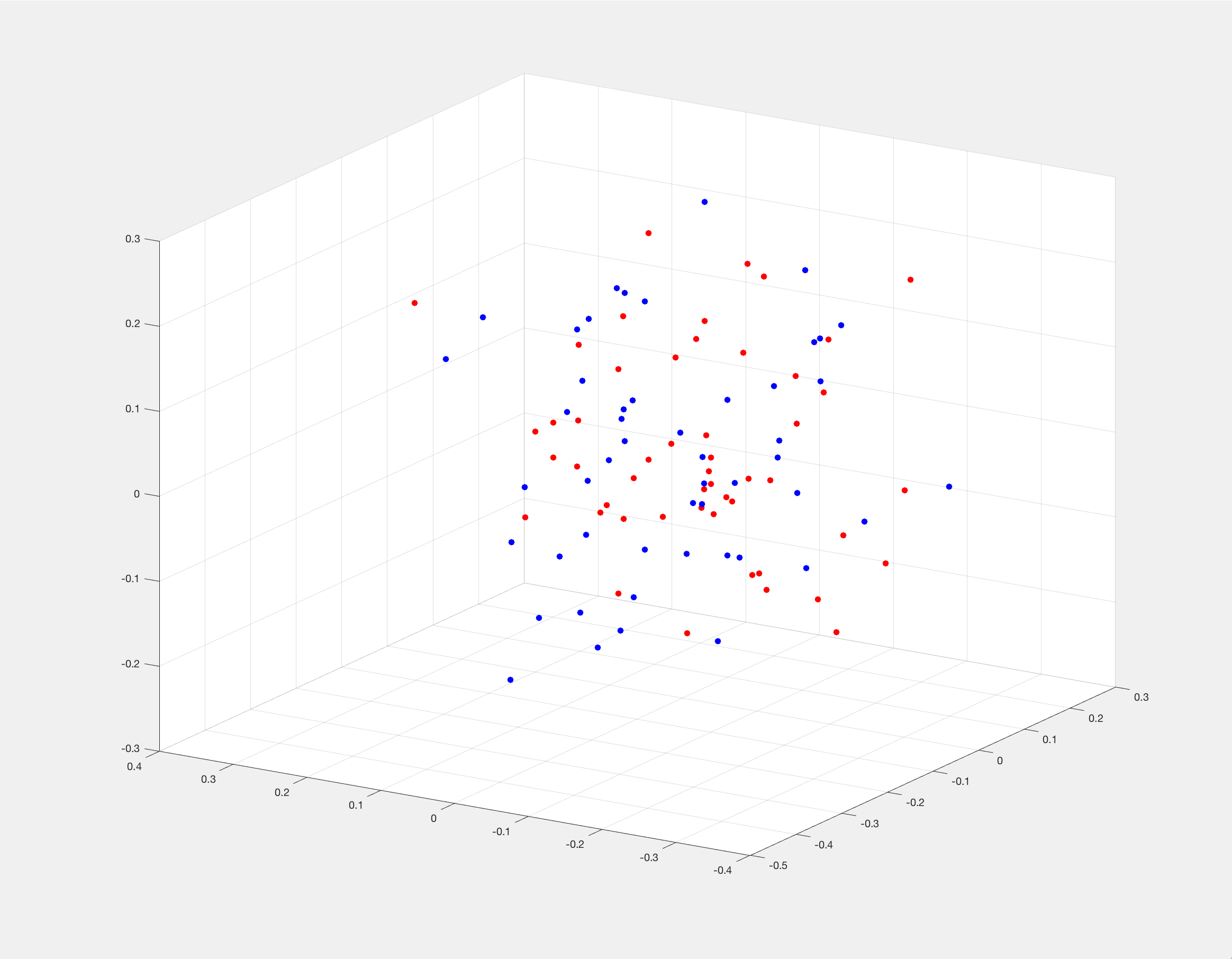
Figure 8 (2 Seconds 2 Norm 3D PCA):

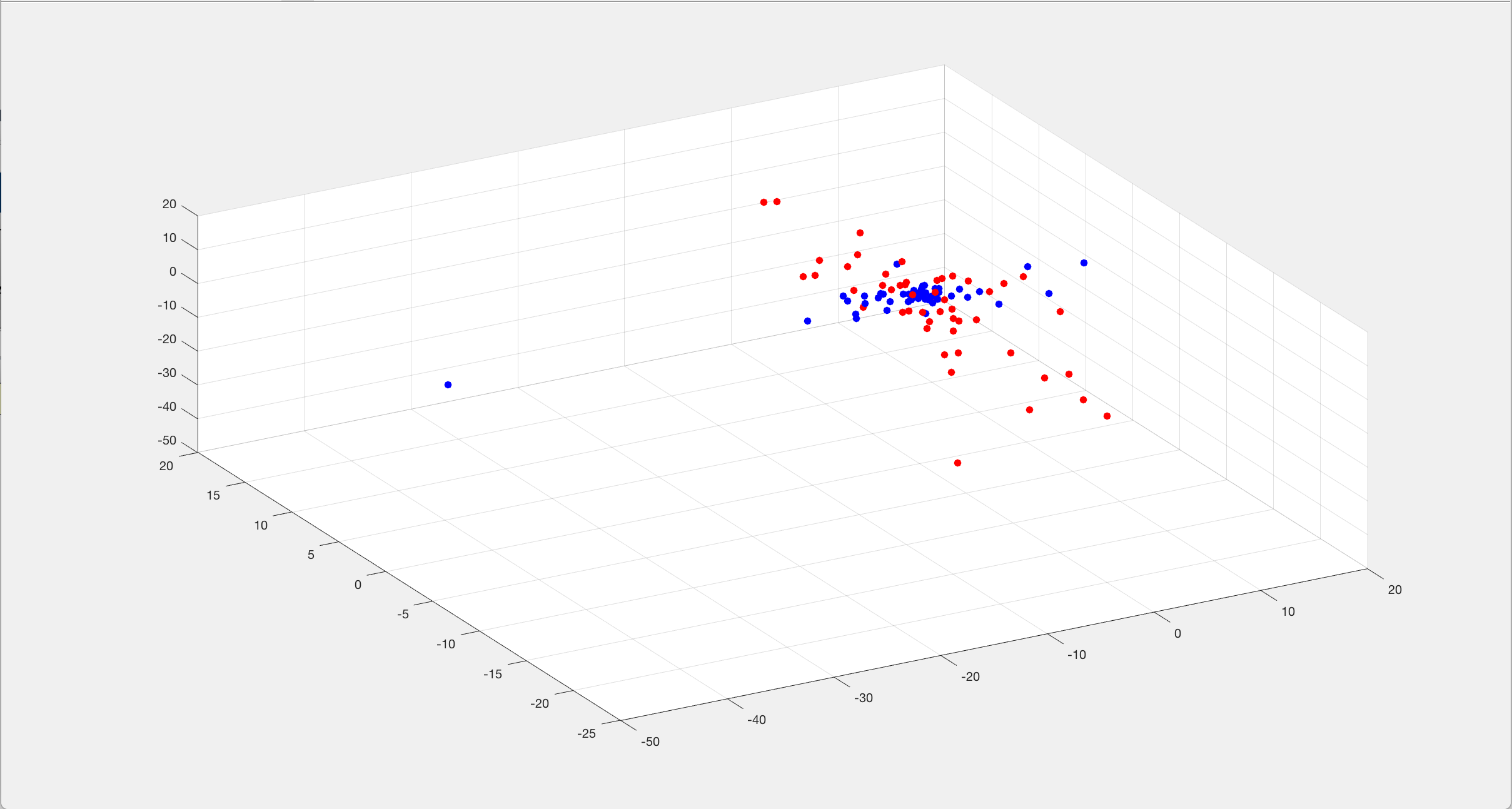
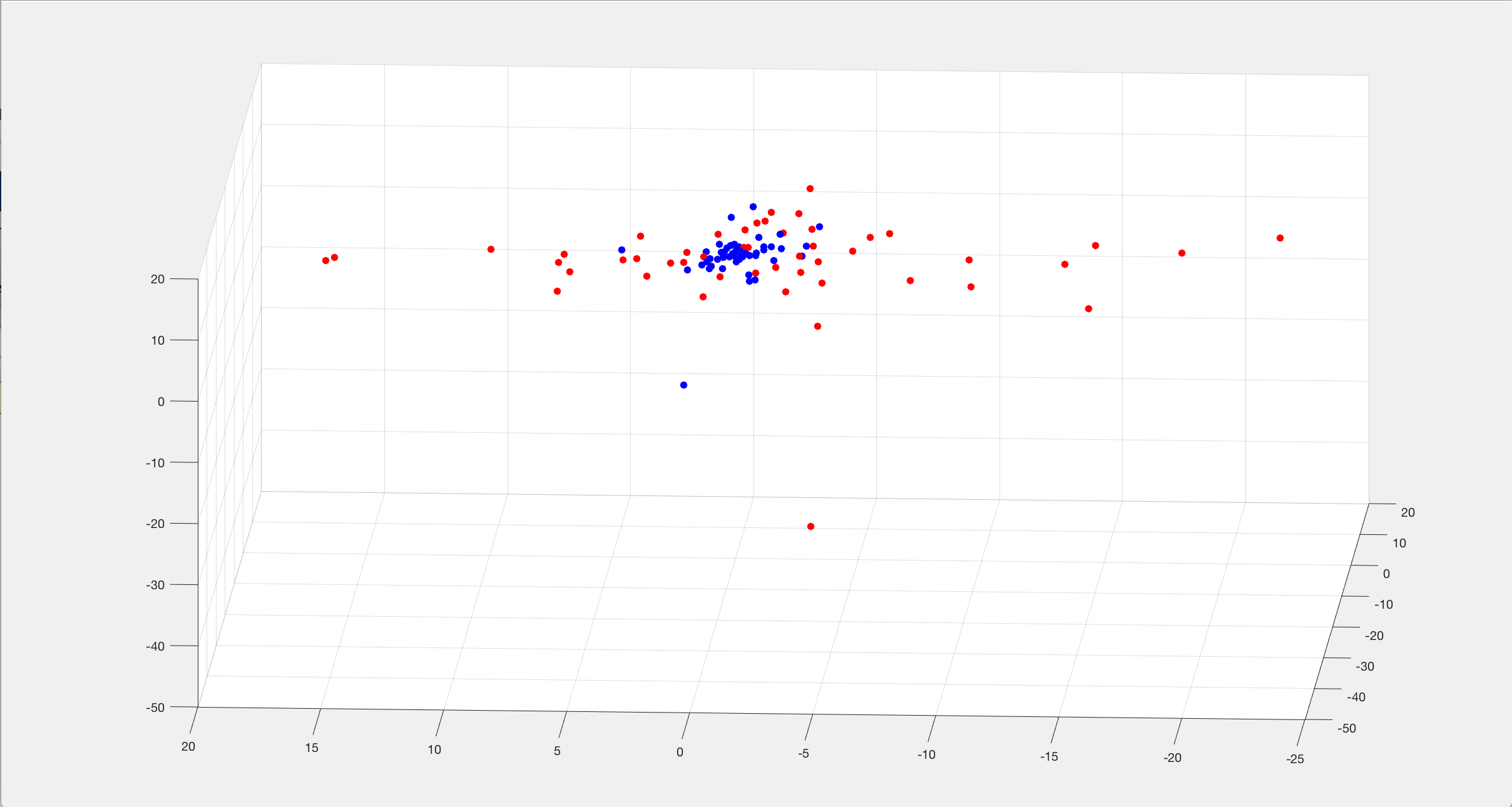
Figure 9 (2 Seconds Infinity Norm 3D PCA):

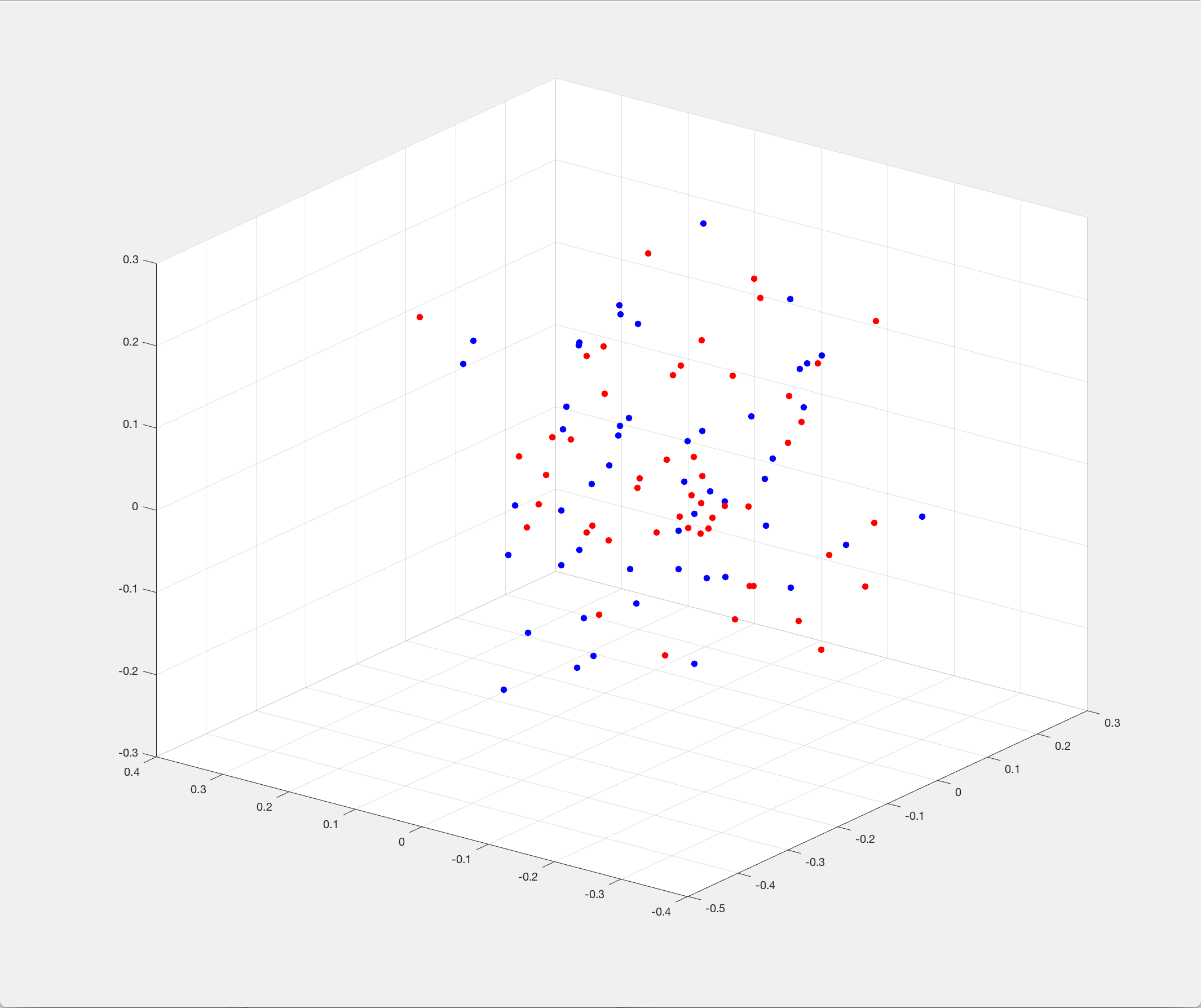
Figure 10 ( 2 Seconds Forbenius Norm 3D):

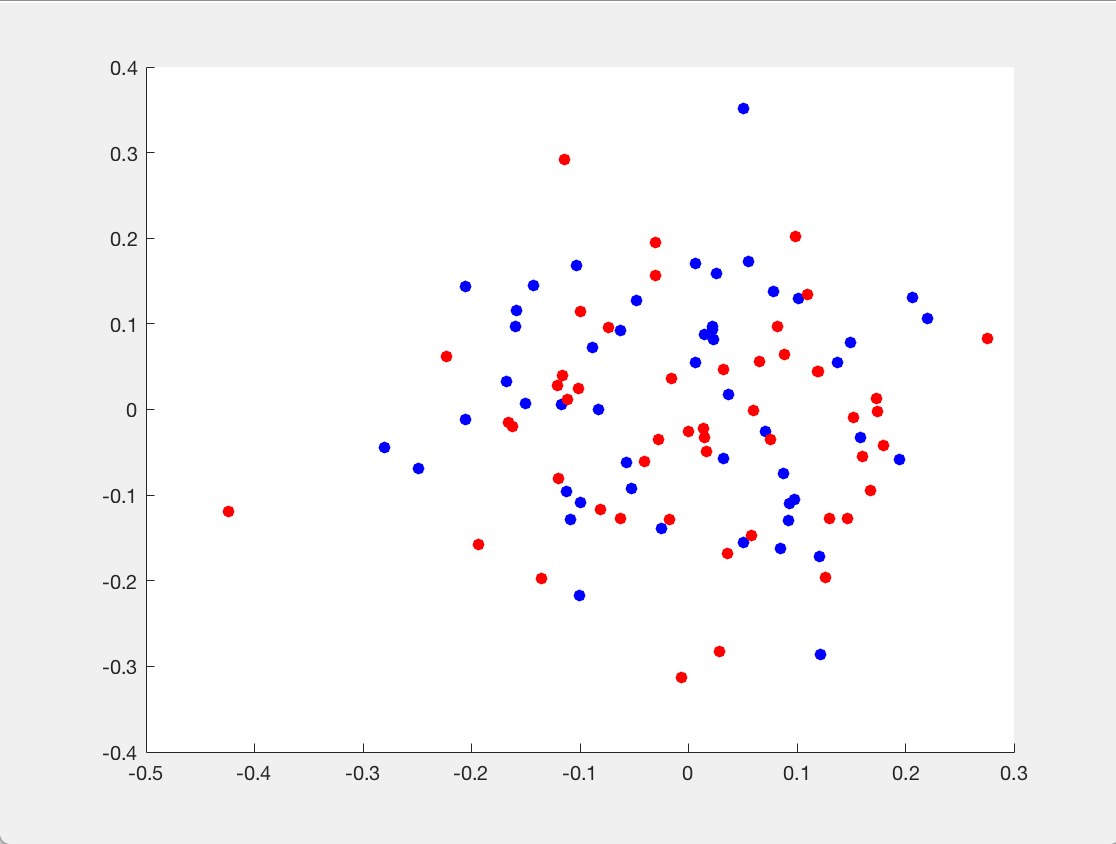
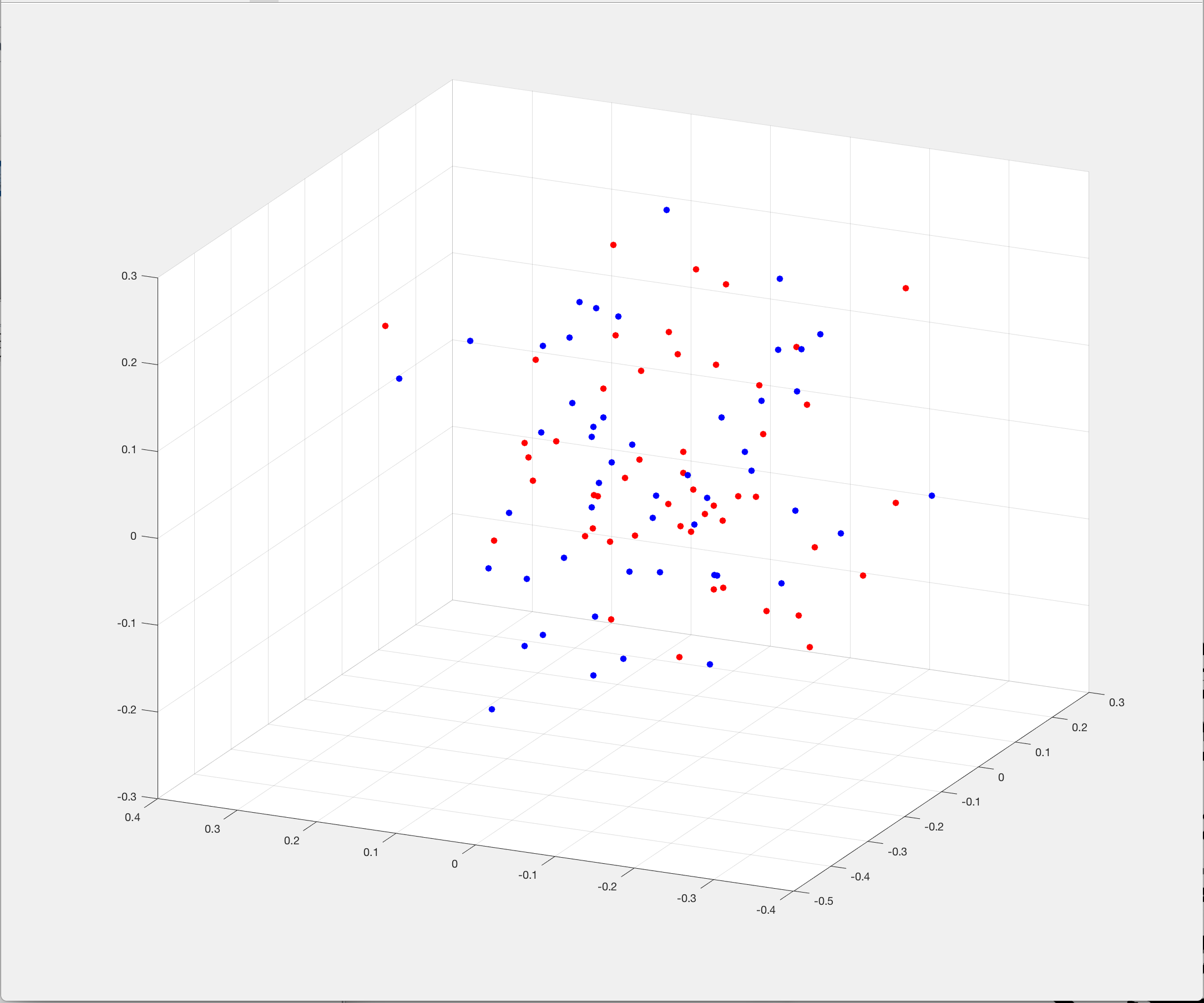
Figure 11a (2 Seconds Fournier Norm 2D): Figure 11b (2 Seconds Fournier Norm 3D):

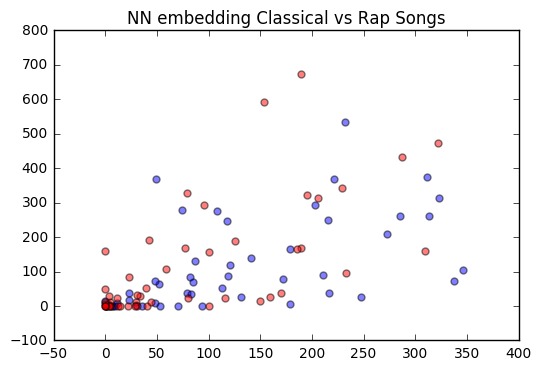
Figure 12 (ReLU Activation Function 2-norm):

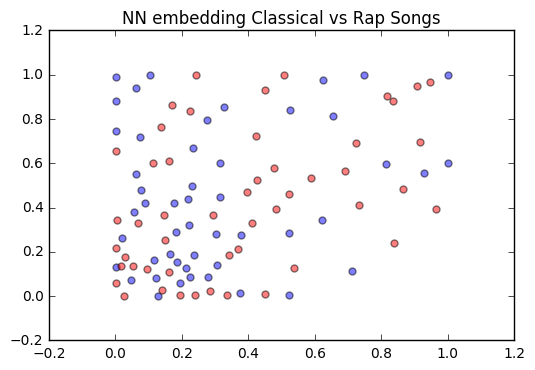
Figure 13 (Sigmoid Activation Function 2-norm):

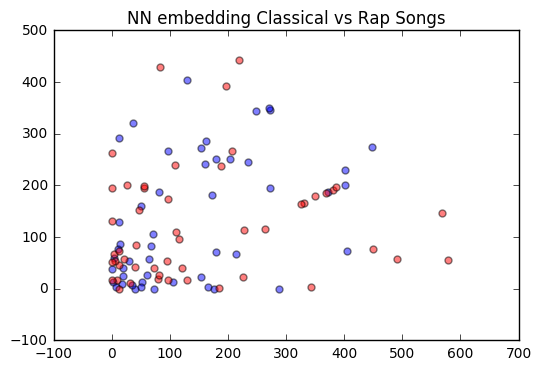
Figure 14 (ReLU Activation Function Infinity Norm):

Table 1 (Neural Net Errors):

|  |  |  |  |
| --- | --- | --- | --- |
| Epochs (by ten thousand) | ReLU (2-Norm) | Sig (2-Norm) | ReLU (Infinity Norm) |
| 0 | 1.001 | 1.001 | 203.420 |
| 1 | .975 | .960 | 202.538 |
| 2 | .63 | .956 | 202.532 |
| 3 | .959 | .953 | 202.527 |
| 4 | .958 | .952 | 202.525 |
| 5 | .957 | .952 | 202.522 |
| 6 | .956 | .951 | 202.522 |
| 7 | .955 | .951 | 202.521 |
| 8 | .955 | .951 | 202.519 |
| 9 | .955 | .950 | 202.518 |
| 10 | .954 | .950 | 202.518 |