THE CLUSTERING PROBLEM

Problem:

Find how many distinct spots are present on the surface of a “brusselator sphere” fig 1.

Details:

The colour gradient represents the value of u = f(x,y,z) where $x^2 + y^2 + z^2 = 1$

There exist about 42,000 data points on the surface of this unit sphere referred to as cpdata (closest point data) which is a ~42,000 x 4 matrix with rows of data that look like: [x, y, z, u]

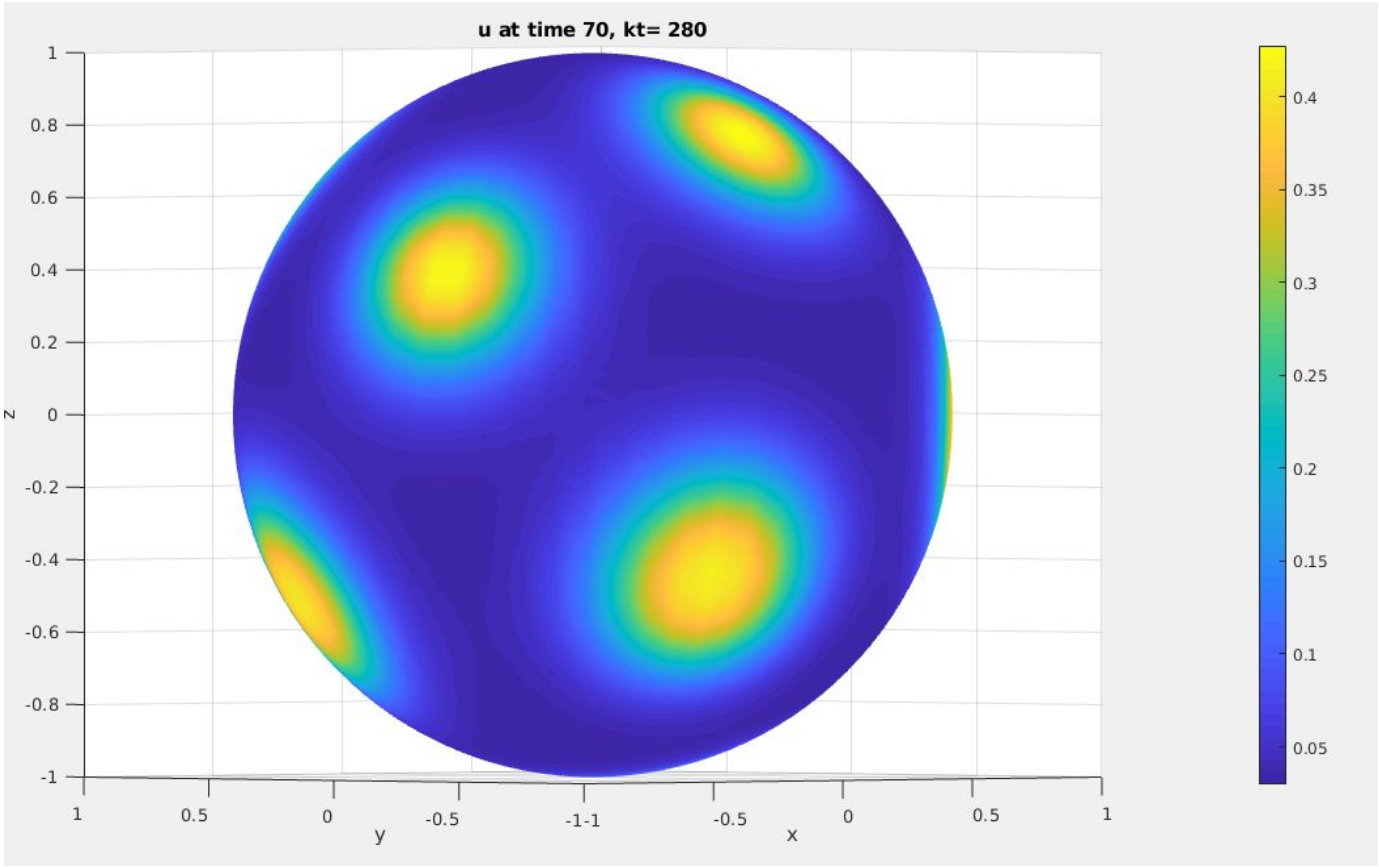


Fig 1: Brusselator Sphere.

The Question:

Filtering this data where u is in a certain range towards the higher end (orange-green) of the "u spectrum" can help us locate ring-like shapes, turning the problem into a clustering problem - fig 2. The question now remains: how do we tell fig 2 contains 9 rings, and therefore 9 spots?

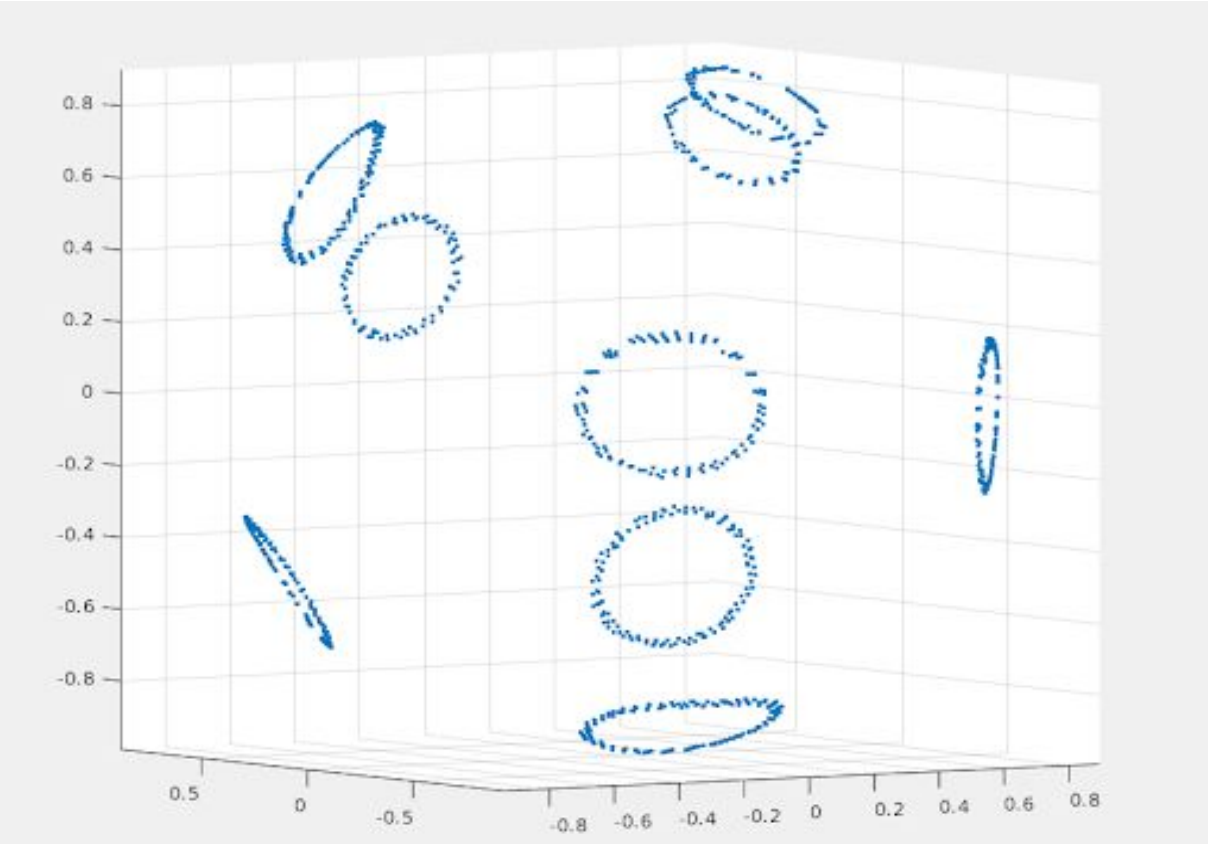


Fig 2: Data Points for which u is in a certain High Range.

Our Ring Algorithm:

One way to solve this problem is to perform a "ring traversal." Starting with any root point (among the "ring" data), we perform a *knnsearch*. We then recruit all the returned data points into a group and then remove them from the ring data. Now we can shift our root point to the farthest neighbour and continue recruiting in a similar fashion until *some* the data points returned by knnsearch are "too far away" to be in the same ring. When this happens, we start a new ring traversal (with a new group). Once we are out of data points in the ring data (or perhaps < 5% of data points remain) we can quit and count the number of ring traversals.

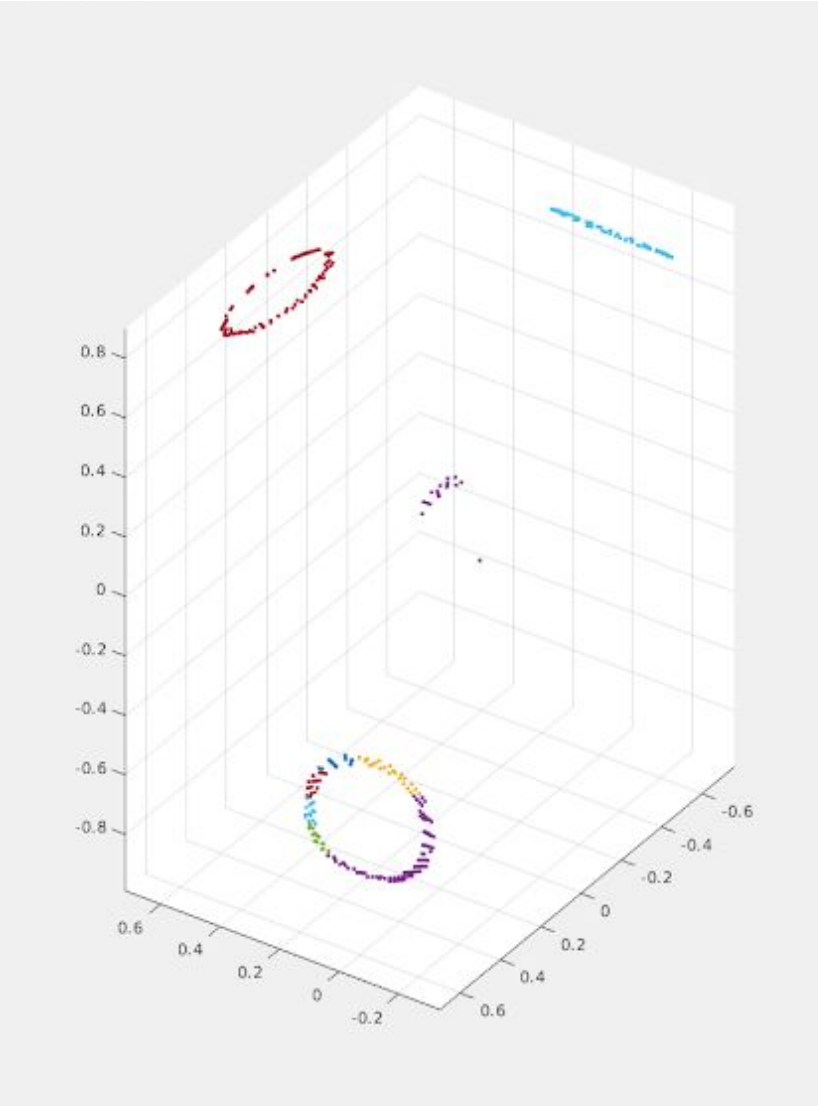
Some parameters we need:

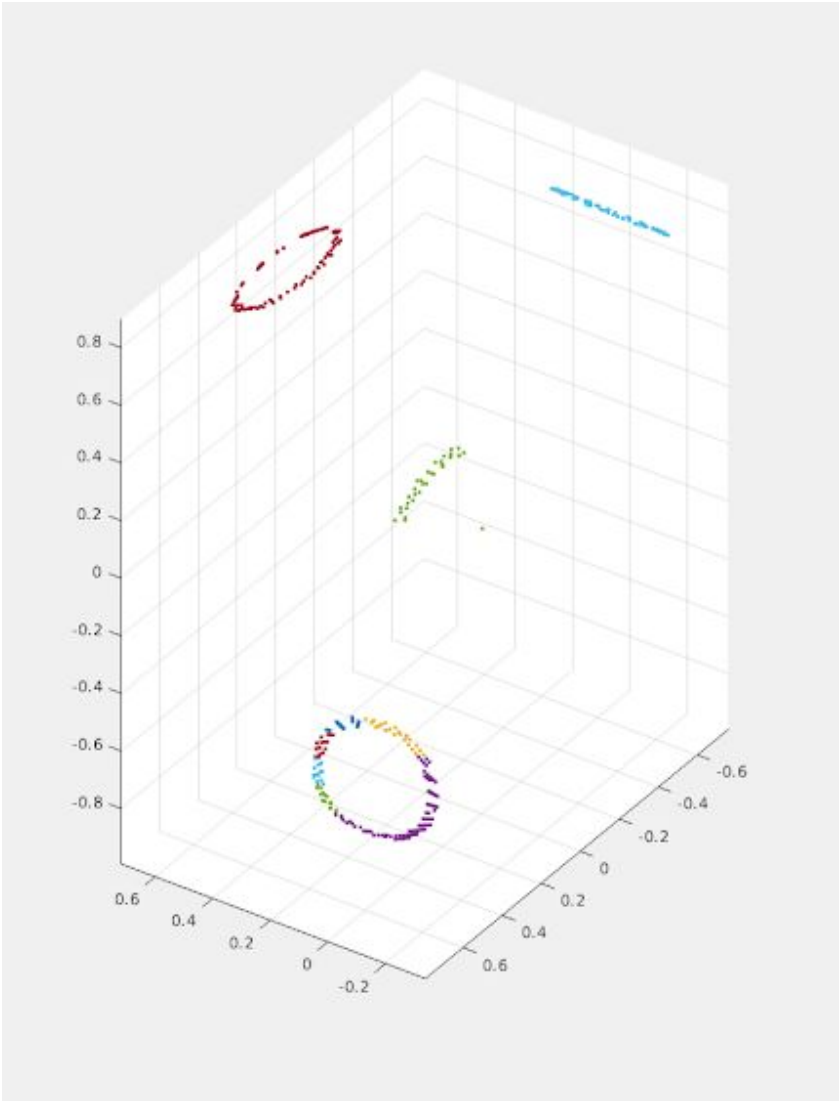
1. U: how high in the "u spectrum" should we be when we filter "ring" data from "raw" closest point data
1. U tolerance (+or- delta U): how many data points should we select for further consideration? That is, how wide should our rings be?
1. K: how many neighbours do we look for
2. Closeness Threshold: how do we decide whether the farthest neighbour belongs to the ring we are traversing or a different ring?

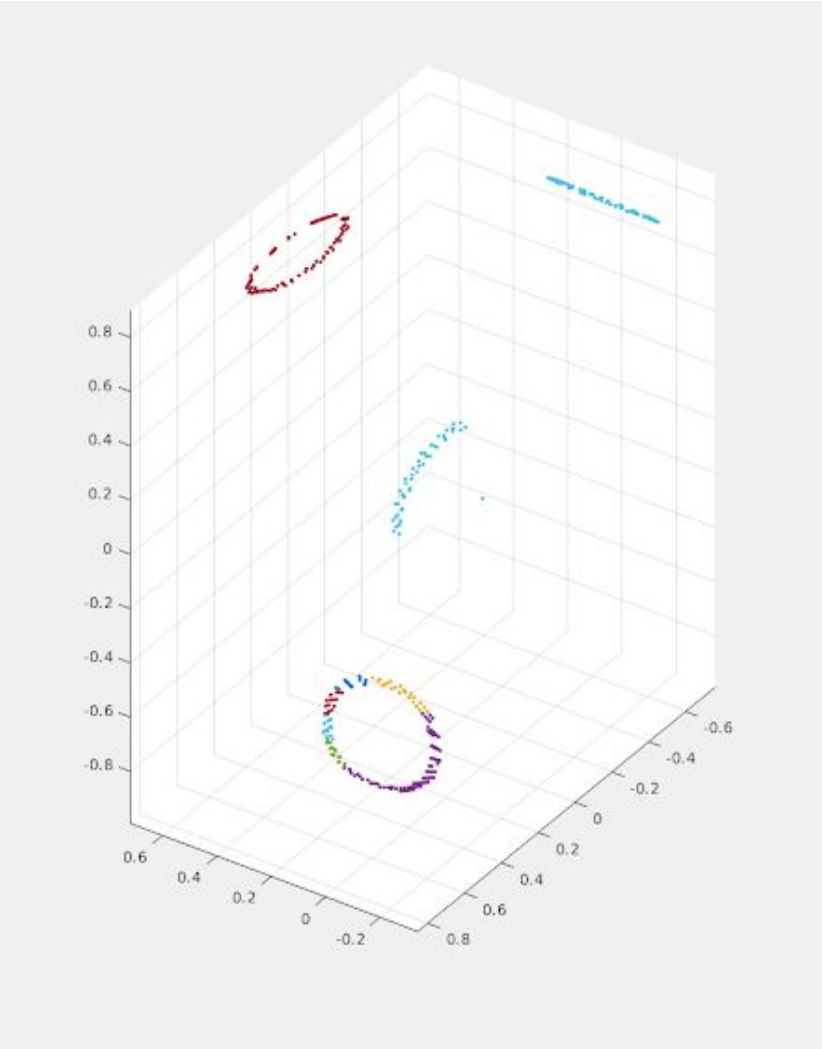
How to choose Ideal parameter combination:

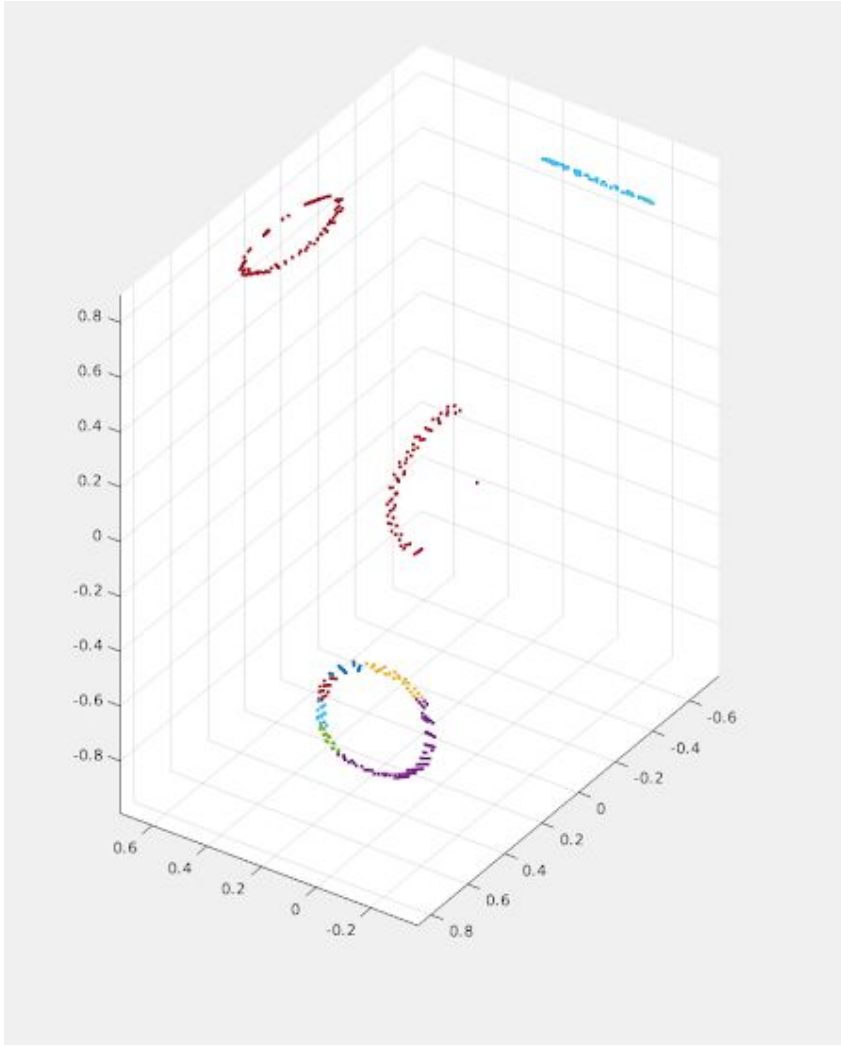
1. U must be in the yellow- orange range such that rings are not too close to each other.
2. U tolerance must be such that the rings are thick enough to be traversed. (note: due to the non uniform nature of "u sample points" on the surface of the brusselator sphere, rings too narrow(i.e. with very low U tolerance) can result in broken rings (fig 7) that are harder to traverse.
3. The wider the rings, the higher should k be, so that, the rings are traversed in a linear fashion. If this is not the case, and k is very small, say 1, we cannot predict "how" the ring would be traversed (certainly not in a unidirectional fashion as we see in figs 3-6). This combined with the closeness threshold can result in a premature termination of a traversal. In other words, we can count a ring multiple times.
4. The higher the value of k, the larger the sections of a ring we scan at a time. Hence closeness threshold must also be higher to allow such scans. But it must be small enough such that data points in a different ring are not mistaken as belonging to the same "section" of our current ring.

Figs 3-6: A Sample Ring Traversal:


0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0.6
0.4
0.2
0
-0.2
-0.6
-0.4
-0.2
0
0.2
0.4
0.6


0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0.6
0.4
0.2
0
-0.2
-0.6
-0.4
-0.2
0
0.2
0.4
0.6


0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0.6
0.4
0.2
0
-0.2
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8


0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0.6
0.4
0.2
0
-0.2
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8

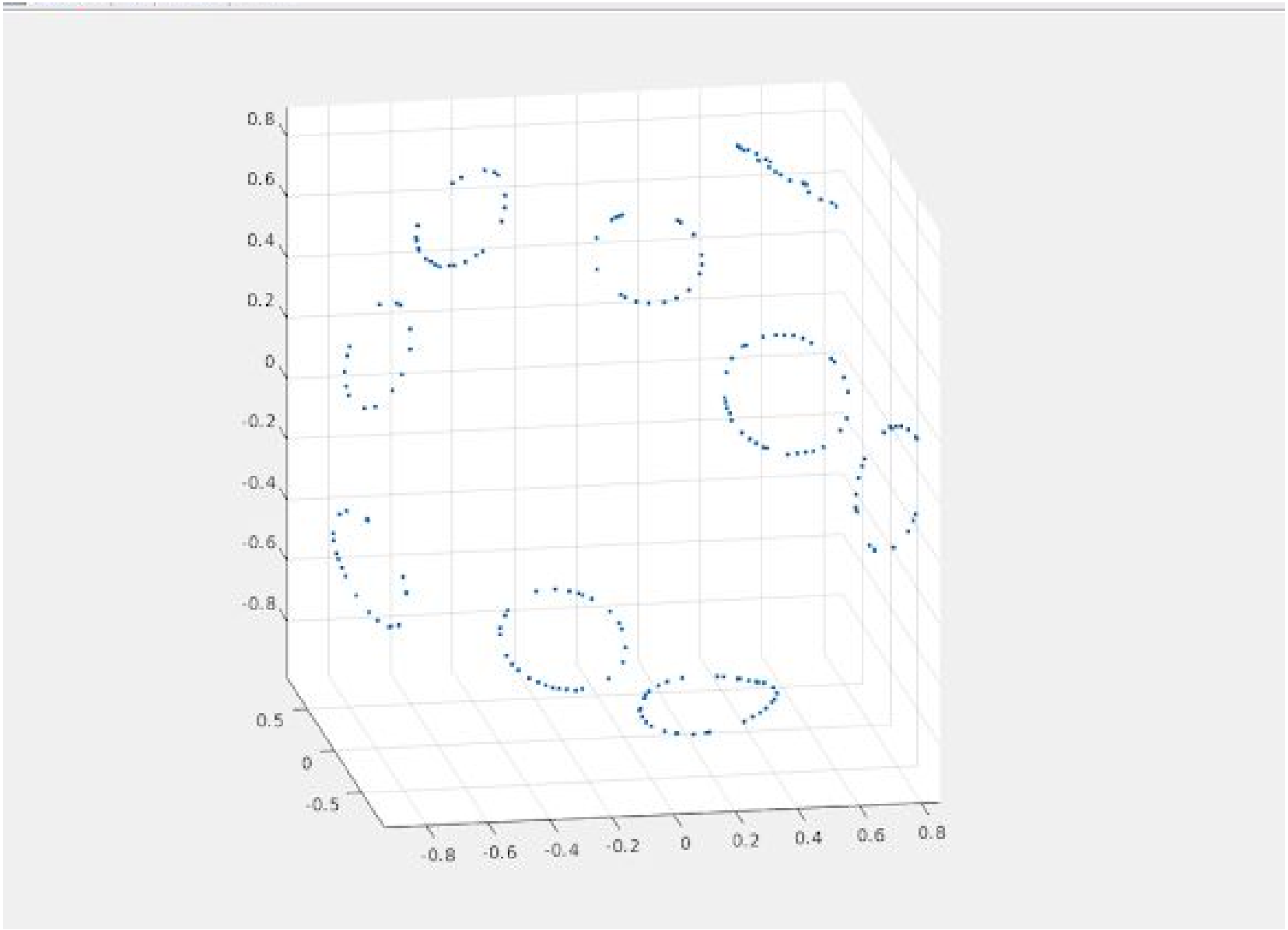


Fig 7: Broken Rings: the result of very small tolerance and therefore very few data points.

Discussion and Questions:

1. This scheme works just fine if the parameters values are ideal (these are mentioned in the code). Nonetheless, this scheme may be totally useless if we do not want to supply these values. It all depends on the context: how far must we go to improve and "smartify" our code? How much self dependent do we want it to be?

2. What happens if a spot are about to split? How do we *even define* when a spot is unique and when they have become 2? This is really hard to define and hence hard to look for.

3. Our ring algorithm can be further modified to analyze the shapes of the found rings. For example, circular, elliptical and dumbbell shaped "rings" can point to splitting spots. Averaging a particular ring's data (or a shape's data in this case) and projecting the result onto the sphere would be very close to the centre of the shape. Then computing the mean/ median "radius" of the "shape" can be used to determine the shape of the spot. Nonetheless, the decisions would be hard because of point 2.

4. The dependence of the ring algorithm on parameters also points to its fragility. A slight twist in the parameters can lead to some points "escaping" the knn scans we perform during ring traversals. This can lead to ridiculous results especially if the "rings" take rare shapes like dumbbell and '8'.

5. There are better methods of course but these are much harder to implement. For example, the "Curdling" algorithm described below would be much more robust, independent of the shapes of spots, and free of a gang of parameters. After reading the curdling algorithm, it may not seem so hard to implement, however there is flaw in this algorithm. I will elaborate on it in the next version of this document. It is related to probability of successful group merges and convex hulls of groups after a certain number of mergers have occurred. The mergers will become progressively harder and after a certain point, even if some groups share a border within the same cluster, it would be very unlikely that they would merge! It turns out that this flaw is fatal and until fixed, renders the method useless and impractical.

The Curdling Algorithm:

a. Filter data points where u > (some threshold) and get clusters of data points instead of rings (fig 8).
b. Assign each data point an additional "group" field and set it to NIL.
c. Randomly shuffle this set of data points and many data points that are "right next to each other" on the sphere's surface, and therefore belong to the same cluster or brusselator spot, are *bound* (with very high probability) to end up right next to each other in our shuffled data array as well.
d. Traverse the data array from start to end and any time 2 consecutive data points are found to be "close[1]," assign them both a *valid* (i.e. unique, non NIL) group number.
e. Shuffle the array again! Traverse it again. Any time 2 consecutive data points are "close[1]", and have a different group number (iii. below comes into play here), do the following:
    i. If exactly one of the points has a valid group number (i.e. not NIL), assign the other point this group number.
    ii. If both points have group number NIL, assign them both a *new valid* group number.
    iii. If both points have valid group numbers that are different, say G1 and G2, make an *entry* in a *side data structure* (such as a list or hash table) that says G1 = G2. (note: this information will be used in future iterations of step e. If we encounter G1 and G2 again, we will not treat them as different groups!). We call this step *a group merge*.

Repeat step e until no *new* group merges occur. The number of *effective groups, as determined by our "side data structure"* is the required answer.

It is worth stopping here for a second to see what is going on in step e. During our traversal, any time iii. (mentioned above) happens, we conclude the following: since some point in G1 is "*adjacent*[2]" to some point in G2, G1 and G2 cannot belong to different clusters in our original problem (fig 8). Hence G1 and G2 are part of the same cluster (a brusselator sphere spot) and the pair of points that established the relationship G1 = G2 *for the first time*, acted as the *bridge* between G1 and G2 *at that time*. Obviously, G′ = G1 U G2 (say) will grow further as iterations of e occur again and again and this *narrow bridge will disappear* as more data points (or groups) *coagulate* onto the *surface*[3] of G′. After sufficient iterations of step e, G′ will eventually grow to form a single group that contains all data points that are present in a cluster in our original problem (fig 8).

---

1. The closeness threshold is very small determined by the average sample point density on the surface of brusselator sphere.
2. Same as 1.
3. A *hull* that surrounds data points in G′

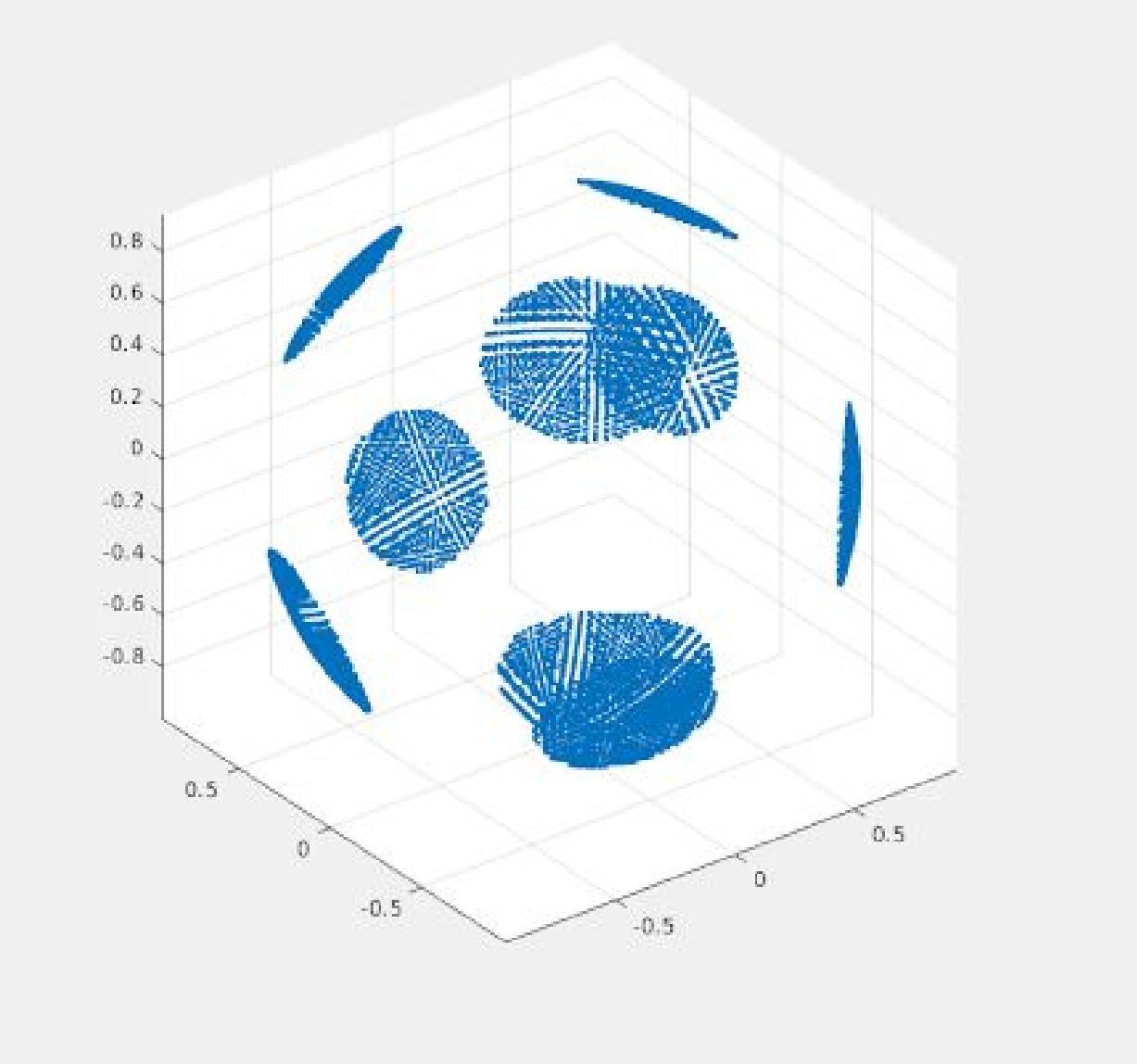


Fig 8: Data Point Clusters.