DEMOCRITUS UNIVERSITY OF THRACE

DEPARTMENT OF MOLECULAR BIOLOGY AND GENETICS

PCA-BASED CLUSTERING OF PROTEIN STRUCTURES

Christos Mantis

Supervisor: Dr. Nicholas M. Glykos

Structural and Computational Biology Lab

Alexandroupolis, Greece

February 2023

Table of Contents

Abstract

Molecular dynamics simulations are used to generate valuable information essential to protein folding. As the algorithms and hardware used for these simulations progress and reach simulation times 10s of μs or higher, they produce large amounts of data that needs to be analyzed. Techniques such as protein clustering, provide a solution to this problem. The goal of this thesis is to develop a fast and efficient computer program for protein clustering. This is achieved through the use of Principal Component Analysis (PCA), a technique based on linear algebra, that allows us to reduce the dimensions of the molecular trajectory data and highlight only the meaningful motions in the simulation. The software was tested using three different peptides and it is able to cluster millions of protein structures in a short amount of time, whilst being memory efficient. This study concludes by suggesting various methods that the code could be improved to cover a larger number of clusters and be more precise with its selection of protein structures for each cluster.

Περίληψη

Οι προσομοιώσεις μοριακής δυναμικής προσφέρουν πολύτιμες πληροφορίες, όσον αφορά την αναδήπλωση των πρωτεϊνων. Οσο ομως οι αλγόριθμοι και οι υπολογιστές που χρησιμοποιούνται βελτιώνονται και φτάνουν χρόνους προσομοίωσης της τάξης των 10μs ή περισσότερο, παράγουν ένα μεγάλο ποσό πληφοριρίας, που πρέπει να αναλυθεί. Τεχνικές, όπως αυτές της ομαδοποίησης πρωτεϊνών, μας δίνουν μία λύση σε αυτό το πρόβλημα. Ο στόχος αυτής της πτυχιακής εργασίας, είναι η ανάπτυξη ενός γρήγορου λογισμικού που θα επιτυγχάνει την ομαδοποίηση πρωτεϊνών. Αυτό επιτυγχάνουμε με τη χρήση της PCA (Ανάλυση Κυρίων Συνιστώσων). Η τεχνική αυτή είναι βασισμένη στη γραμμική άλγεβρα και επιτρέπει τη συμπήκνωση των δεδομένων μοριακών τροχιακών και αναδικνύει μόνο σημαντικές μταβολές που συμβαίνουν κατά την προσομοίωση μοριακής δυναμικής. Το λογισμικό δοκιμάστηκε με τρεία διαφορετικά πεπτίδια και έχει τη δυνατότητα να ομαδοποιεί εκκατομύρια δομές σε μικρό χρονικό διάστημα, ενώ ταυτόχρονα είναι αποδοτικό ως προς τηνεξοικονόμηση της μνήμης του συστήματος. Αυτή η εργασία ολοκληρώνεται με έναν αριθμό προτάσεων που θα μπορούσαν να βελτιώσουν το λογισμικό, όπως η δυνατότητα να ομαδοποιεί περισσότερες ομάδες πρωτεϊνών και να είναι πιο ακριβές στην επιλλογή των πρωτεϊνικών δομών κάθε ομάδας.

Key words

Principal Components Analysis (PCA)

Protein Clustering

Molecular Dynamics (MD)

Variance

Standard Deviation (SD)

1. Introduction

1.1 Protein Clustering

Protein clustering is a method used to extract useful information and analyze data from molecular dynamics simulations. A large amount of trajectory data is produced from such simulations, which makes the need for clustering tools all the more imminent [1][2]. The technique used for this thesis derives from PCA and is based on the comparison of the principal components of the dataset.

1.2 Principal Components Analysis

Principal Component Analysis (PCA) is a technique used to highlighting patterns in a given dataset. It works by reducing the dimension of said dataset, via calculating its eigenvectors and their corresponding eigenvalues [3][4]. PCA has a multitude of uses in image compression, face recognition software and more. In our instance we use PCA to calculate the differences between protein structures.

1.3 PCA In Protein Clustering

As mentioned previously, longer simulations mean more trajectory data. The way PCA tackles this problem is by only taking important motions the molecule makes into consideration, instead of each individual atomic trajectory [5]. Dimensionality reduction, also proves to be advantageous when it comes building fast and memory efficient software. Lastly PCA is based on linear algebra, which computers have an easier time calculating [6]. All these points, give PCA an edge compared to other clustering methods, in the speed and efficiency department.

1.4 C Programming Language

As we said before, we want this algorithm to be fast and efficient. This means that we need a programming language that can make a lot of calculations per second, without running into errors, and provides control over memory.

C was the obvious choice for this project because it combines all the elements we mentioned above. It can run on almost all systems and provides manual control over memory, allowing for

greater optimization of the code to run faster and be less taxing on the computer's resources, by only allocating the appropriate amount.

1.5 How we tested the software

In order to test the software, we gathered a dataset of proteins. Three proteins were used for the purpose of this thesis. First was the CLN025 peptide, which was used to develop the code. After it was finished, two more peptides were used to test its functionality. These are the peptide T and the DNA binding peptide. Results from the clustering of all 3 peptides will be presented and discussed later.

1.6 Visualization of the results

Finally, to help give a meaning to our numerical results, we used three pieces of software to visualize it. The R programming language allowed us to plot data and discuss our next steps while building the software, as well as portray our results. For the molecules, we used carma64 [7][8] to create the superposition data and RasMol [9] to visualize it.

2. PCA Clustering Software

The first question raised while developing the software, is how do we use the data given to us and extract protein clusters from it?

The input we use is a file that contains frames of the molecular dynamics simulation, along with the first principal components that describe each frame. These principal components are the most important ones, and we can use them to determine how similar or different two protein structures are. The way we do that is by calculating the Euclidean distance between the respective components of each two frames.

2.1 Using the Principal Components to Calculate the Variance

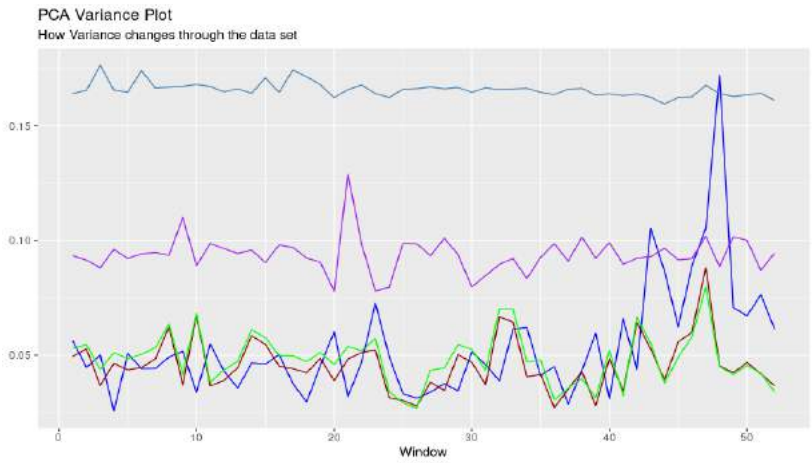
Now that we know what these principal components tell us about each structure, we can use this information to extract the clusters. The first step, is to calculate the variance of the Euclidean distance between a set number of frames. The number of frames used to calculate this variance

will be referred to as window from now on. For example, if we have a window of 100 frames, we calculate the variance for frames 1-100. Then the window shifts by 1 frame and

calculates the variance for frames 2-101. This process continues until we reach the end of the file.

Now we can plot the results by having the window in the x axis and its corresponding variance in the y-axis. What we expect to see is a fluctuating line. This line shows us if the structures in a given window are similar to each other or not. If they are, then the variance should have a low value. In theory, in the lowest points of the line, we expect to have very similar protein structures, which are our clusters.

This, of course, varies with different window sizes. If the window is too small, then it is difficult to identify the clusters, because of the high affinity of neighboring frames. Too big and the variance will not fluctuate enough. The next task then, is to find a fitting and standardized window size.



Window	Color
200	Blue
400	Dark Red
500	Green
700	Purple
900	Steel Blue

Figure 1: Change of Variance in different window sizes.

In Figure 1 we can see this fluctuating line in action. Each line represents a different window size. All of these were run in the same software and hardware, with the only difference being how many frames were used to calculate the variance.

After reviewing the results in Figure 1, the number 200 was chosen as the constant window size, amongst all the peptides. The reasoning being, that it appears to have the great consistency in the identification of clusters, whilst retaining a small window value. This means that we have fewer principal components to compare, therefore making the software faster and be more memory efficient. Later in this thesis, we are also going to suggest a potentially better way of calculating the number of frames to calculate the variance.

2.2 From variance to clusters

After reviewing the variance plot, we run into another problem. How do we identify a cluster and its contents from these low variance points? To solve this, we need to find a cutoff value. Frames with distance values lower than the cutoff should belong in the same cluster.

When it comes to the cutoff, we decided to test out standard deviation (SD), as it is easily calculated from the variance we already possess. We simply take the square root of the variance, and we have our cutoff. From this point forward, we are going to refer to this as σ-cutoff.

2.3 Finding the first cluster

We first have to find out which window has the lowest variance. Then we calculate the σ-cutoff using the variance of that point. Then we calculate the Euclidean distance between the reference frame (the first frame of the window) and all the other frames of the dataset. If the Euclidean distance is lower than the σ-cutoff, then that frame belongs to said cluster. The number of the frames that are identified as part of the cluster and their corresponding principal components are then saved to a matrix, to be later printed as the output. This process is then repeated with different σ-cutoffs, these being multiples of SD.

2.4 Optimizing the software

As it has already been mentioned, one of the main advantages of this software is its speed. To optimize it, is to make it run calculations in parallel via multithreading. To do so, we can split the data evenly amongst available threads, using the p_thread library. After splitting the data, we then calculate the all the parameters for calculating variance in the first window of each thread.

These parameters are the distances between the principal components of all the frames of the window and how many those are.

When it comes to the calculation of variance, we use two shortcuts to improve runtimes. The first is the use of a single pass algorithm to compute variance [10]. The other has to do with the means in which we obtain the values to compute the variance in each window.

For the purposes of this example let's assume we have a window size of 200. The software scans through the frames in a sliding box fashion. The first window calculates the Euclidean distances between frames 1-200. After the variance for this window is calculated, the window then shifts by one frame and now covers frames 2-201. Instead of calculating all the Euclidean distances from the start, we can just get rid of the ones that have to do with the first frame of the previous window. In this case this is frame 1. After subtracting those values, we can then add the values from the new frame, that being frame 201. By doing so we do not have to compute the all the Euclidean distances from the start, effectively skipping a few steps with zero repercussions.

When it comes to memory optimization, we utilize the malloc function. The software calculates the number of values needed for each of its matrices and uses malloc to allocate the appropriate amount of memory, making sure to free said memory as soon as it is done with it. All the optimizations mentioned and more can be viewed in greater detail in the source code part of the thesis.

3. Results

Three peptides were used to test the software, the first of which was also used for its creation. For the purposes of this thesis, the software was optimized to fit the input data of each of the peptides. This means that certain functions were changed in order to fit the number of principal components.

The data used for the clustering of these three peptides was provided by [15][16][17].

3.1 CLN025 Peptide

CLN025 is a peptide that consists of 10 residues. It derives from the C-terminal region of protein G and is useful because of the unique structure it folds into, while in room temperature [11][15].

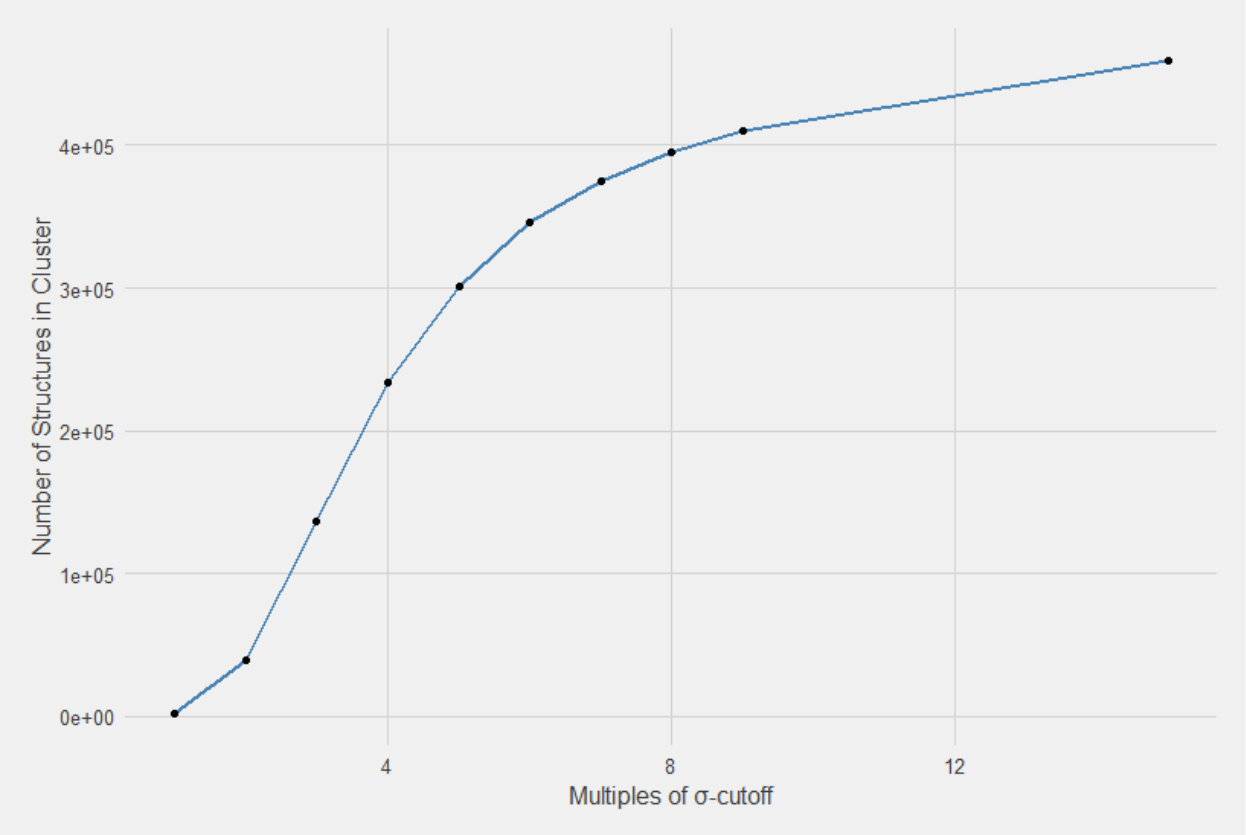


Figure 2: Number of structures in the first cluster per multiple σ-cutoff for CLN025 peptide.

As expected, after increasing the σ-cutoff value, the number of structures in the cluster also increases drastically. The question then is, how big can we make the σ-cutoff value, before structures that do not overlap with the other start being added to the cluster?

To answer this question, we visualized the structures in the RasMol software tool and via a PC (Principal Component) plot. In Figure 3 we can see the structures of the first cluster, as we increase the value of the σ-cutoff.

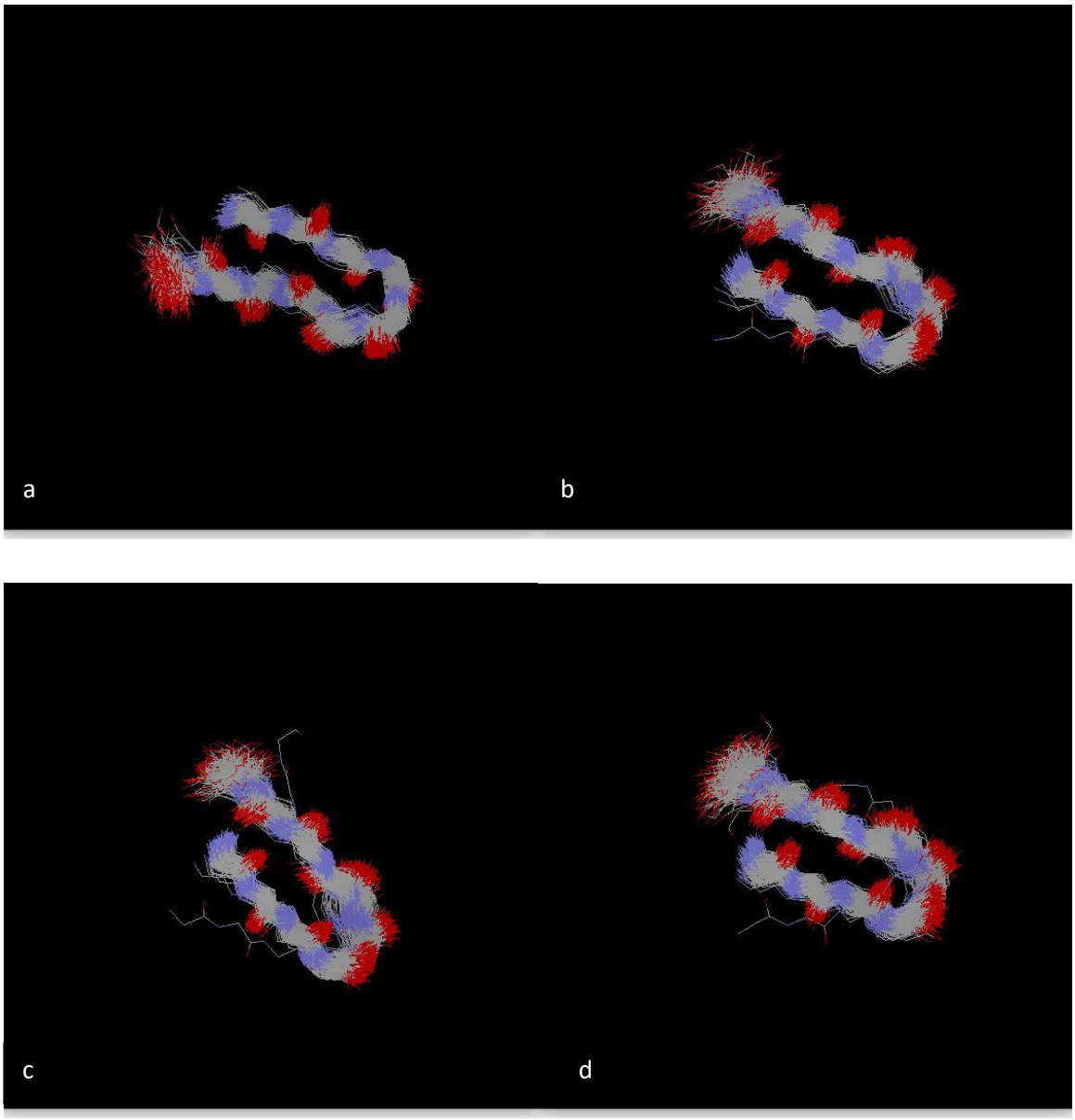


Figure 3: Superposition of 500 structures with increasing σ-cutoff from the first cluster of CLN025 peptide.

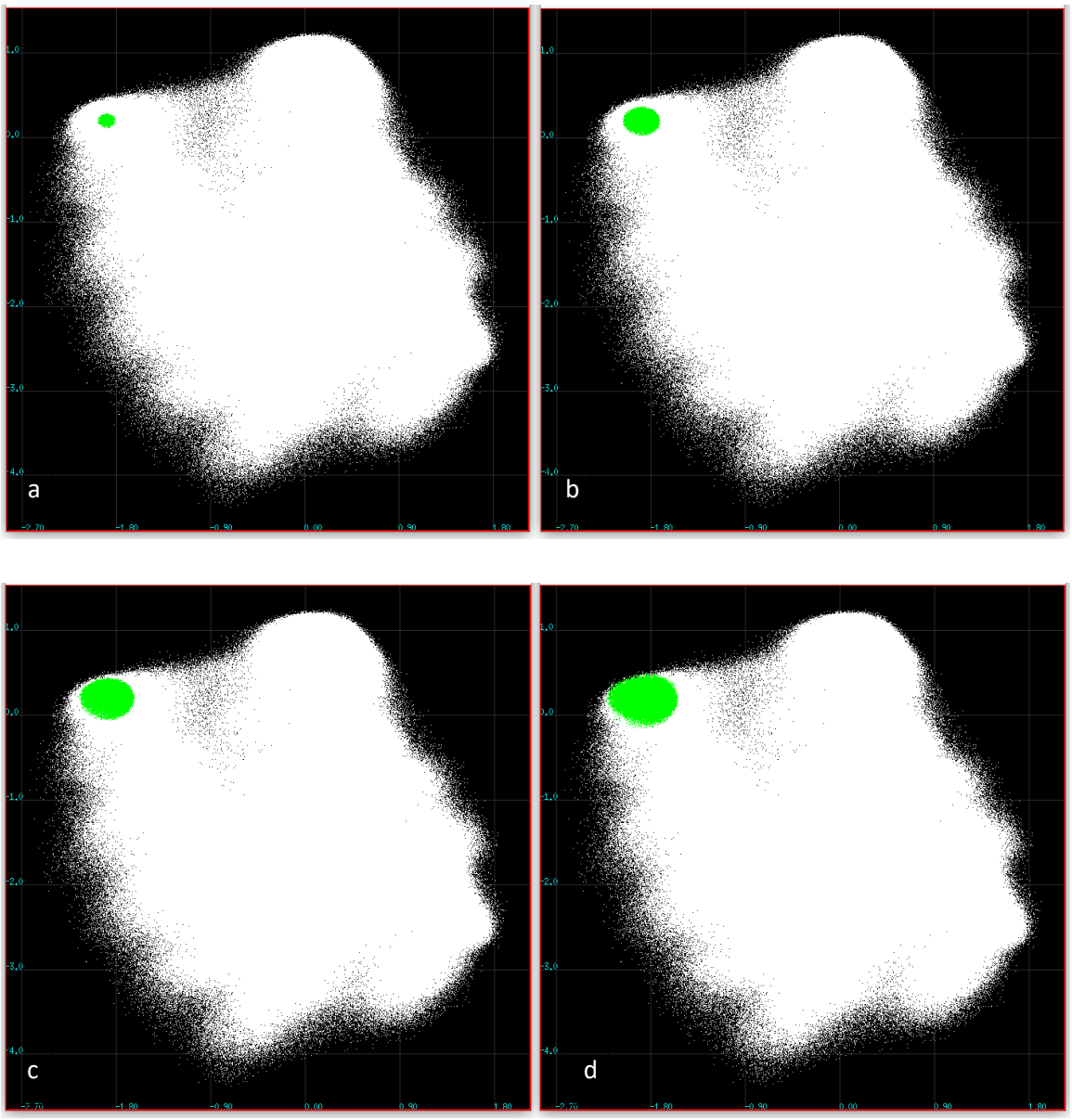


Figure 4: Scatter plot of the first two Principal Components of CLN025 peptide, showing the first cluster with increasing σ-cutoff.

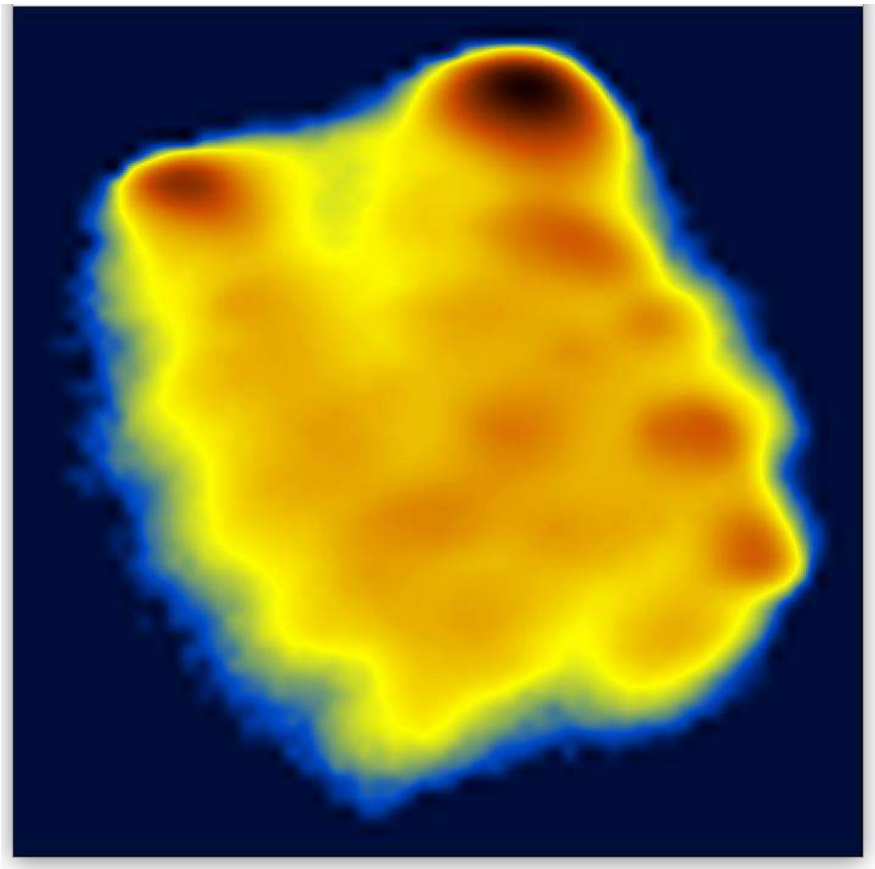


Figure 5: Logarithm of the PC1/PC2 density distribution of CLN025 peptide.

For the CLN025 peptide we can see the location of the first cluster in Figure 4, compared to the log distance matrix in Figure 5. After we reach a σ-cutoff of 2, we start seeing non overlapping structures, as evident in Figure 3(c). In general, we can see that as the σ-cutoff increases the structures still stay mostly similar. This is best seen in Figure 3(a), whereas more significant differences start appearing in Figure 3(d).

3.2 Peptide T

Peptide T is a fragment of gp120, the coat protein of the HIV virus[16]. The fragment consists of residues 185-192 and is able to inhibit the binding of isolated gp120 and HIV-1 to the CD4 receptors in vitro [12].

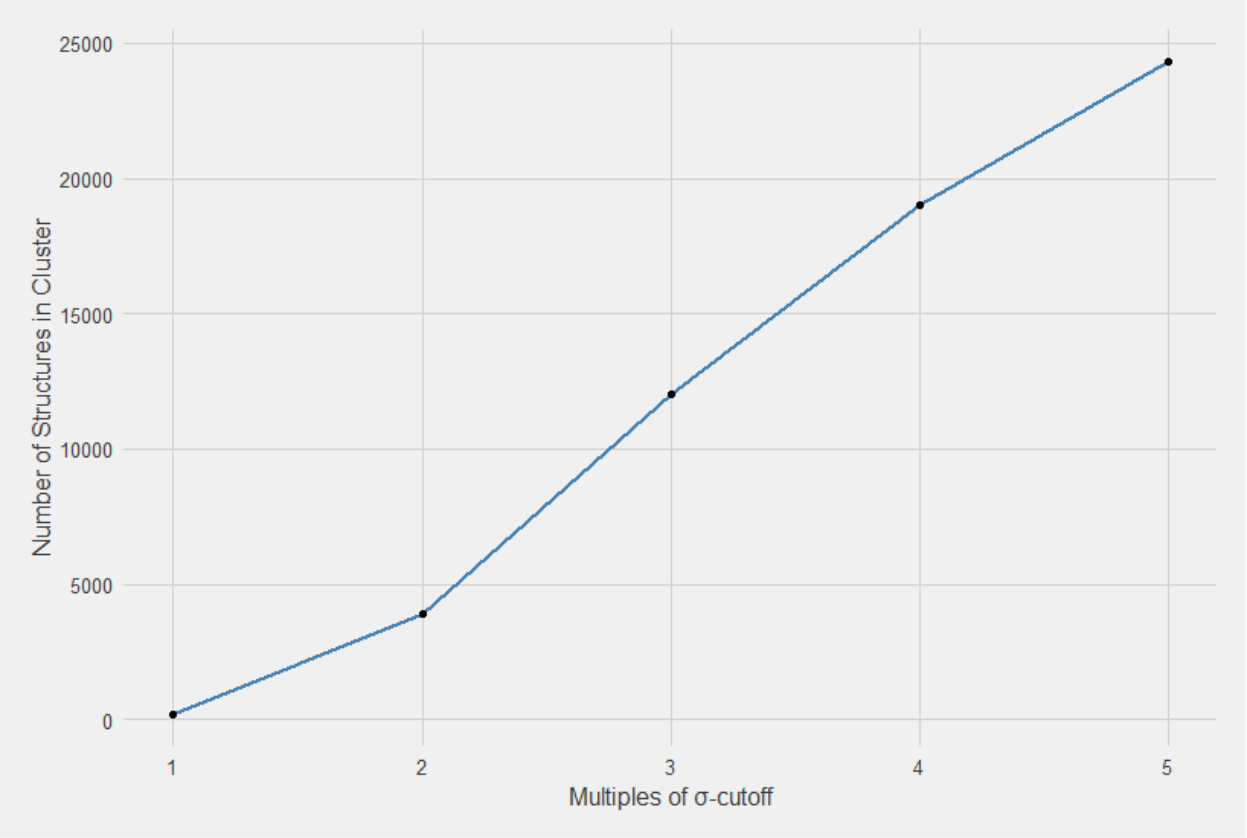


Figure 6: Number of structures in first cluster per multiple σ-cutoff for peptide T.

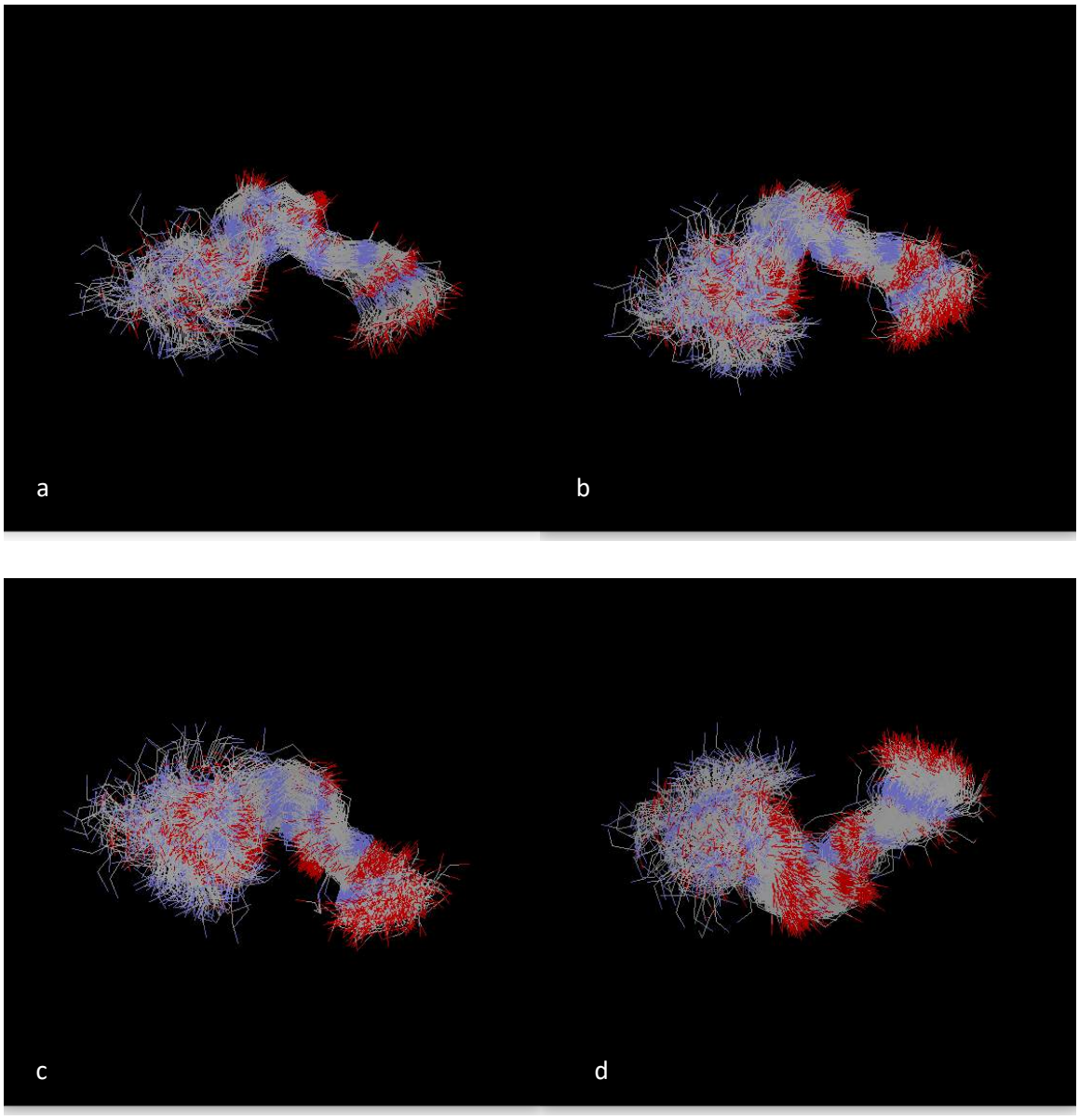


Figure 7: Superposition of 500 structures with increasing σ-cutoff from the first cluster of peptide T.

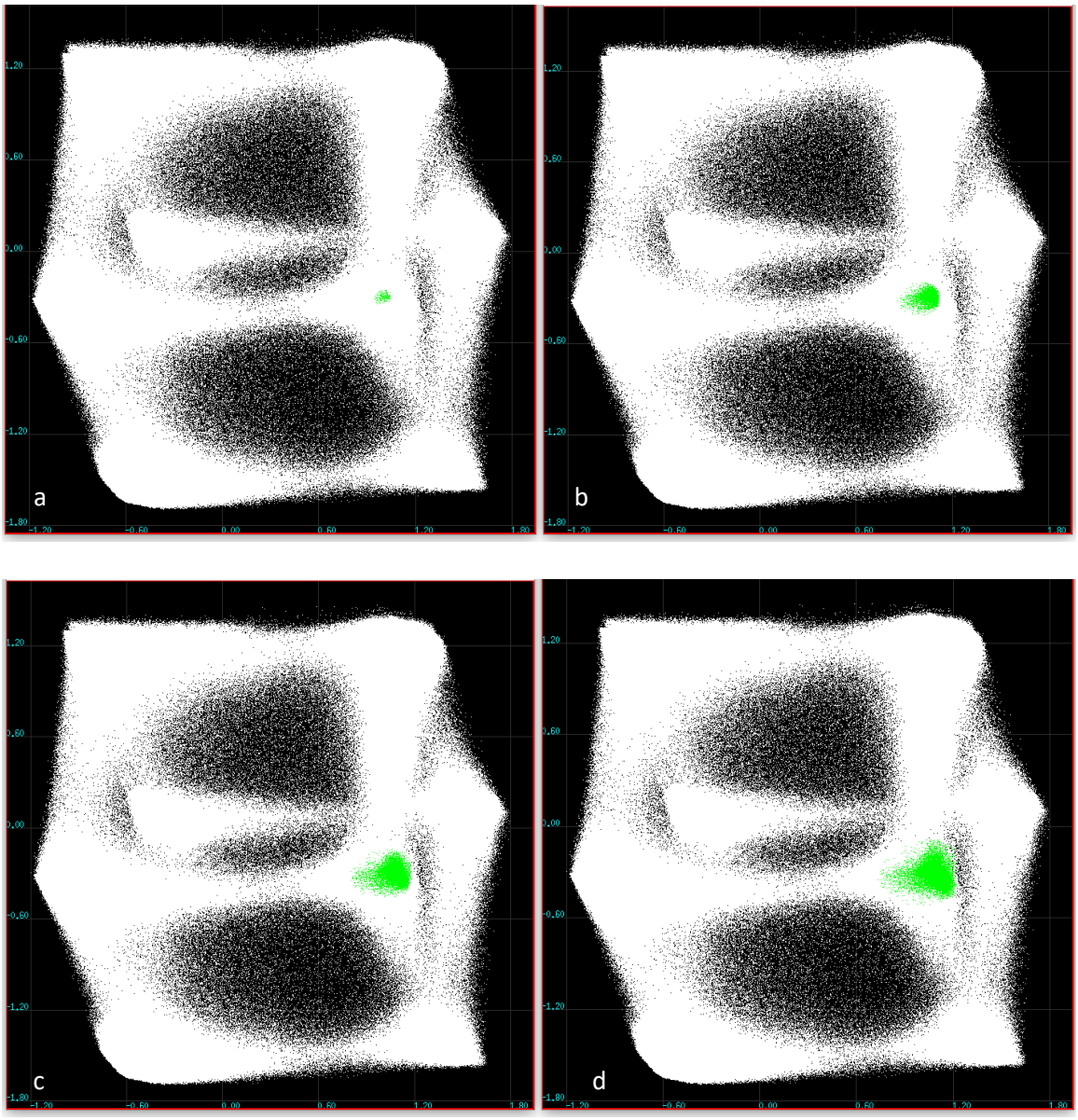


Figure 8: Scatter plot of the first two Principal Components of peptide T, showing the first cluster with increasing σ-cutoff.

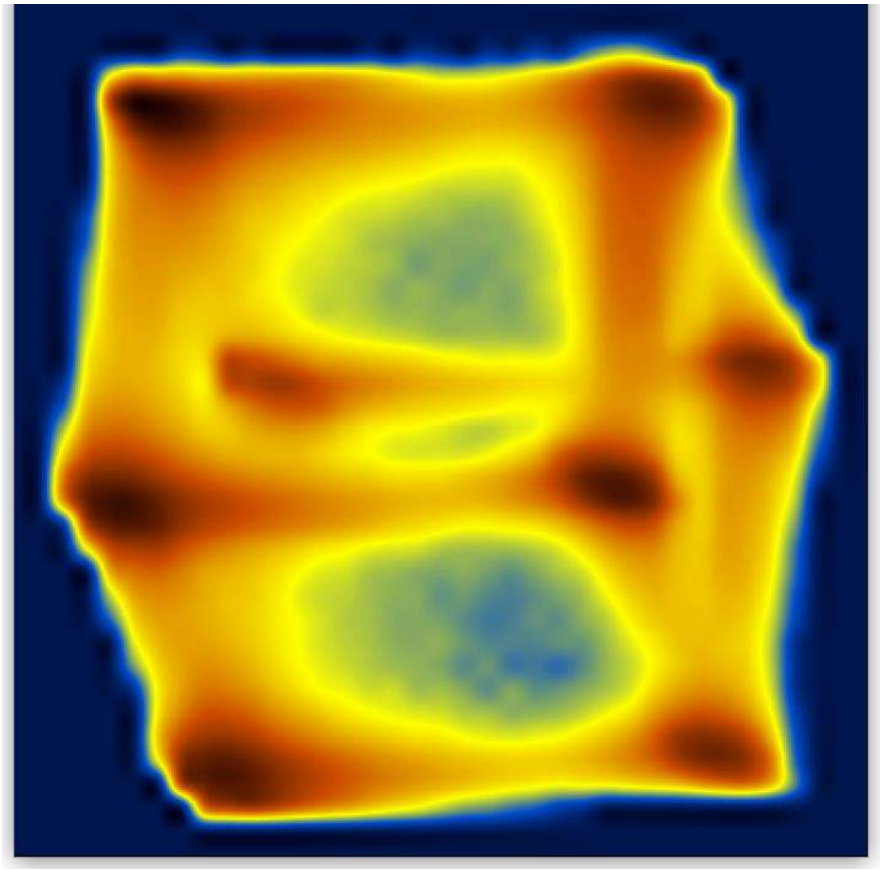


Figure 9: Logarithm of the PC1/PC2 density distribution of CLN025 peptide of peptide T.

Out of all three peptides, peptide T has the most linear increase in structures per σ-cutoff. After comparing the structures in Figure 7, there do not appear to be significant differences between the four iterations of the first cluster, which means that the σ-cutoff, might have been able to be higher.

3.3 DNA Binding Peptide

A 12-residue peptide that can bind to DNA [13][17]. DNAbp also has high affinity for other organic and inorganic molecules, like Gallium nitride (GaN) and Hydroxyapatite [14]. From this fact we can hypothesize that this peptide might have more than one stable tertiary structure.

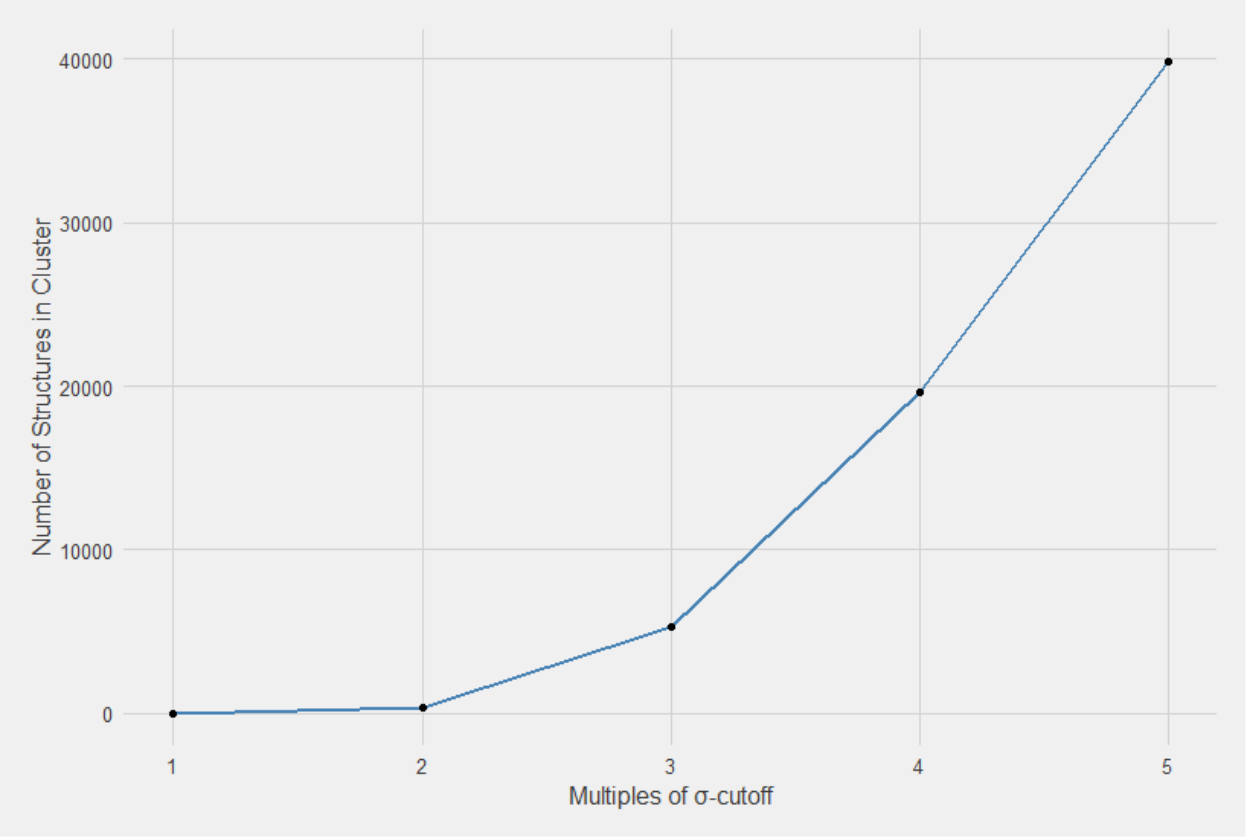


Figure 10: Number of structures in first cluster per multiple σ-cutoff for DNA binding peptide.

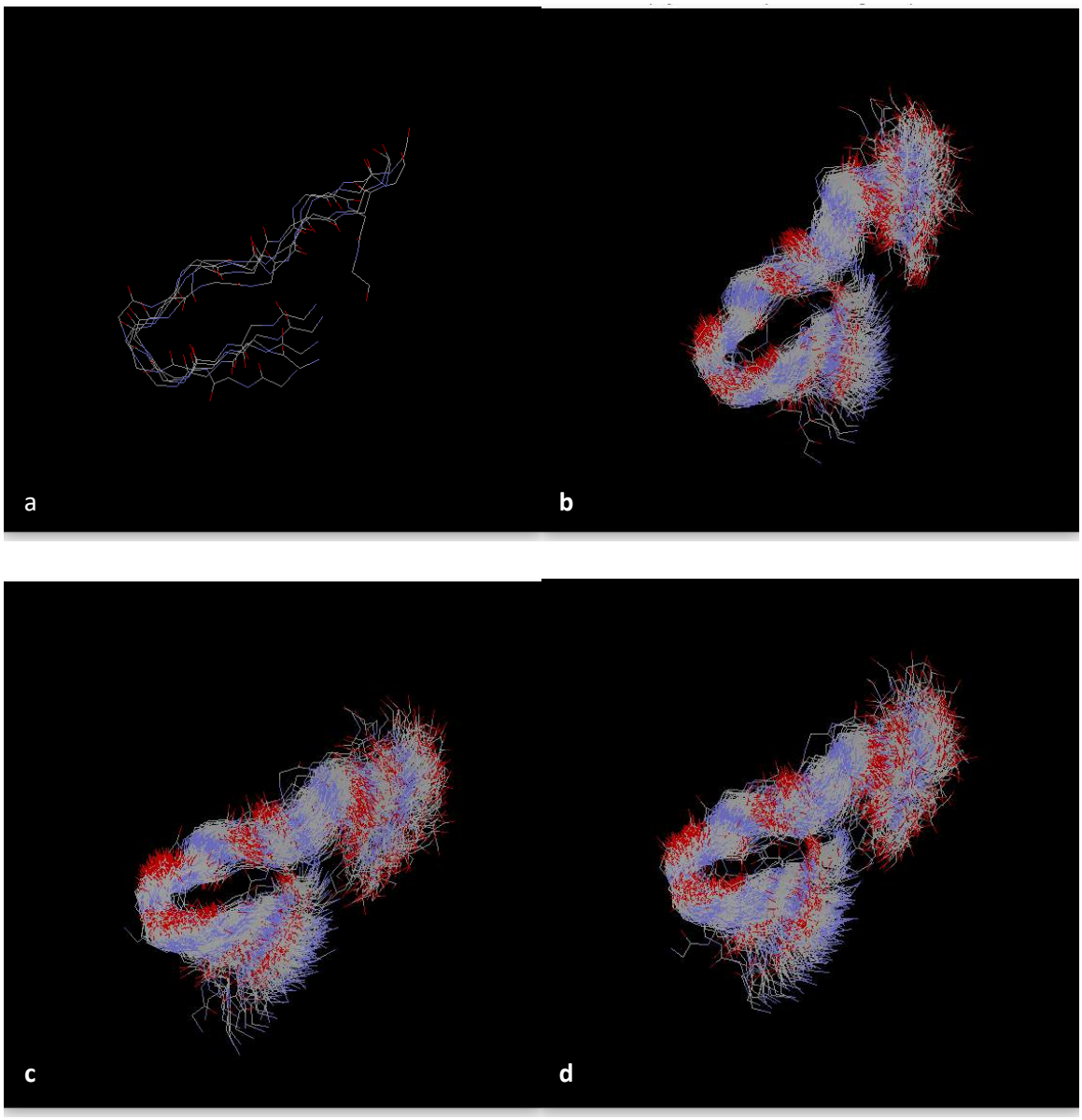


Figure 11: Superposition of 500 structures with increasing σ-cutoff from the first cluster of DNA Binding Peptide.

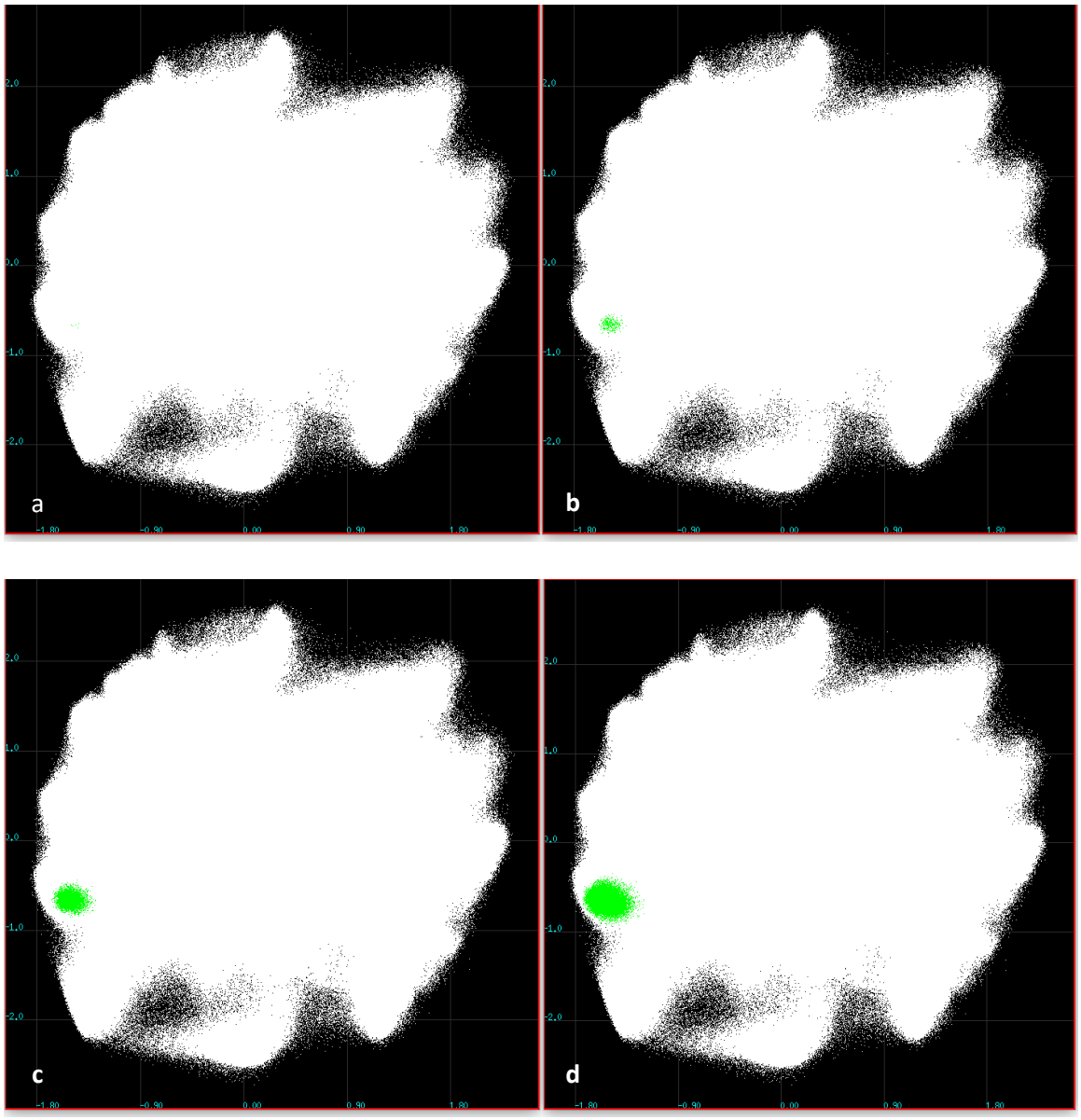


Figure 12: Scatter plot of the first two Principal Components of DNA binding peptide, showing the first cluster with increasing σ-cutoff.

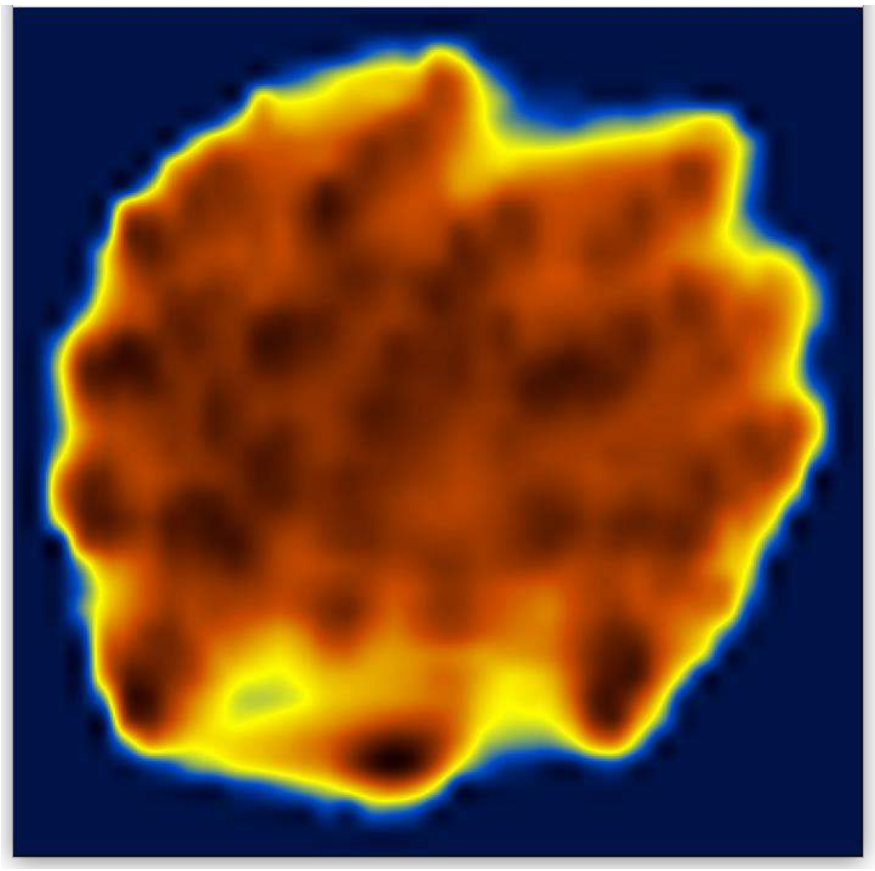


Figure 13: Logarithm of the PC1/PC2 density distribution of CLN025 peptide of DNA binding peptide.

Compared to the other two peptides, this one has the lowest number of structures associated with the cluster at a σ-cutoff of 1. Although, if we increase that σ-cutoff, it appears that the number of structures added to the cluster increases exponentially. Furthermore, we expect that as the cluster gets bigger, the structures portrayed in Figure 11 would start to have significant differences, however that does not appear to be the case, up until Figure 11(d).

4. Discussion

4.1 Conclusion

After testing the software with the three peptides, we can safely arrive at the conclusion that it can indeed compute the first cluster of the dataset. This means if the algorithm develops to the point where it can compute more than one cluster, effectively, PCA-based clustering has the

potential of being a more mainstream method of clustering. However, we are still not at that point yet. We still have not found a concrete computational method in determining where a cluster ends and where another one starts. The following part will be dedicated to making suggestions on the next steps to fix some of the problems that occurred after the review of the results of the software.

4.2 Improvements to the algorithm

It is now clear that there are obvious improvements that can be made to increase its functionalities and make for a better workflow. We are going to suggest two major upgrades and one quality of life one to the software.

4.2.1 Number of clusters computed

One of those suggestions has to do with the number of clusters the software can compute. Currently the code only accounts for the first cluster found in the data. The computation of more clusters is not as easy as, find the next lowest variance point, compute the SD and compare principal components to find the members of said cluster. The problem lies in the fact that two or more lowest variance points might belong in the same cluster. The solution is to integrate an algorithm to determine whether two or more clusters are the same and merge them.

The integration of this algorithm is not an easy task. If done incorrectly it can have a big impact in the computation speed of the software and slow it down significantly. If implemented correctly, it severely increases the capabilities of this software, making PCA based clustering of protein structures one of the most efficient and high-speed methods of protein clustering.

4.2.2 Alternative frame window computation method

The other has to do with its accuracy and the number of frames used to calculate the variance in each window. In this thesis, we used the value 200 as the constant number of frames. We believe there is a better way to compute this value. This method involves computing the max Euclidean distance from all the frames, with a sampling factor of 1. The process would then be repeated, until the sampling factor reaches a high enough number, after which the max distance values reach a plateau and would not fluctuate in a meaningful way.

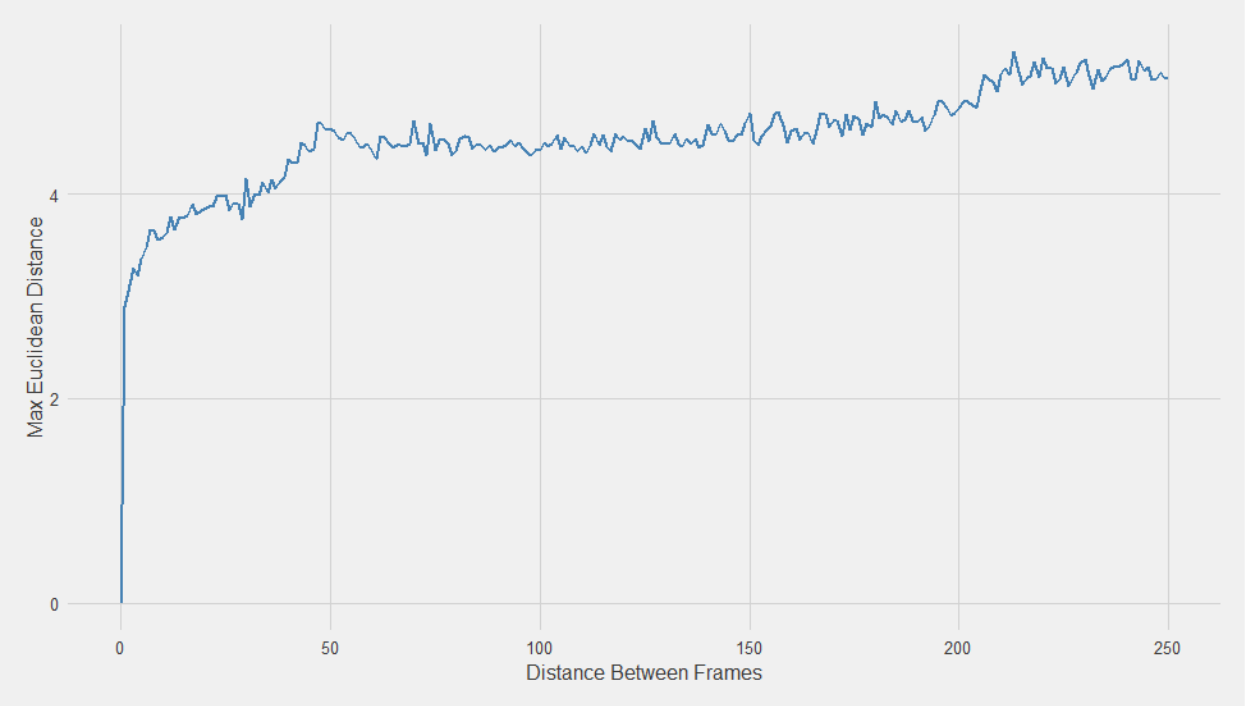


Figure 14: Max Euclidean distance vs sampling factor

After plotting the results, as seen in Figure 14, we then have to find the point when the curve reaches the plateau and get the value for the window size. This method has some obvious upsides and downsides. In theory its accuracy surpasses the one we currently use in this thesis. On the other hand, this would make it harder for the computer to do all the calculations necessary to determine this value, slowing down the whole process. Then there is also the problem of different peptides having different points in which they reach the plateau, which would make the algorithm even more complicated.

As a whole this method has some accuracy benefits, but it might prove to be more time consuming. It needs to be tested to determine if it is more fitting and should replace the current version.

4.2.3 Adjusting for the number principal components

The last improvement we are going to suggest in this thesis is a quality of life one. Integrating a system that allows for the automatic adjustment of the software, based on the number of principal components would significantly improve its workflow. This must be done without

slowing down the software, for example by using more nesting for loops. However, this is a minor improvement compared to the abovementioned, which significantly impact the functionality of the software.

5. Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <pthread.h>

#define WINDOW 200  //Window = how many frames will be used to calculate variance
#define THREADS 4   //Number of threads used should be changed depending on system specs

void *thread( void *input);


int NumberofLines, mills;
float** Data_Array;
float* Variance_Matrix;

int main( int argc, char *argv[] )
{

int i, h, j, min_frame = 0;
char number[60];
float min_var;

   //Exit program if there is no input file

if( argc == 1 )
      {
      printf("No input file was given\n\n");
      exit(1);
      }

   //Open input file and assign pointer to it
   //Get the number of lines in the file

FILE *fp;

   fp = fopen(argv[1], "r");
```

```
   fseek(fp, -74, SEEK_END);

   fscanf(fp, "%d", &NumberofLines);

if(WINDOW > NumberofLines)
   {
   printf("Window given cannot be used\n\n");
   exit(1);
   }

   //Allocate memory for data array

Data_Array = malloc(sizeof(float*)* NumberofLines);
for( i = 0; i < NumberofLines; i++)
      {
      Data_Array[i] = malloc(sizeof(float) * 6);
      }

   //Load input file to memory

fseek(fp, +0, SEEK_SET);

   for(i = 0; i < NumberofLines; i++)
      {
      for(j = 0; j < 6; j++)
         {
         fscanf(fp, "%f", &Data_Array[i][j]);
         (fp, +1, SEEK_CUR);
         }
      }
   fclose(fp);

   //Allocate memory for variance matrix

Variance_Matrix = malloc(sizeof(float)* (NumberofLines-WINDOW));

   //Create threads and split workload amongst them

mills = NumberofLines/THREADS;

pthread_t th[THREADS];

for (i = 0; i < THREADS; i++)
```

```
    {
    int* fraction = malloc(sizeof(int));
    *fraction = i+1;
    if (pthread_create(&th[i], NULL, &thread, fraction) != 0)
        {
        perror("Failed to created thread");
        }
    }

    //Join all threads

for (i = 0; i < THREADS; i++)
    {
    if (pthread_join(th[i], NULL) != 0)
        {
        perror("Failed to join thread");
        }
    }

    //Find the minimum variance and save the window it belongs to

min_var = Variance_Matrix[0];

for(i = 0; i < NumberofLines - WINDOW; i++)
    {
    if( Variance_Matrix[i] < min_var)
        {
        min_frame = i;
        min_var = Variance_Matrix[i];
        }
    }

    //Calculate standard deviation and compare the algebraic distance between princilan
components
    //of min variance frame and all other frames available
    //If the distance is smaller than the standard deviation, it belongs to the first cluster
    //Save the frame number and its components in the cluster matrix to print it as the output

float Distance;
float sigma;
int cluster[1000000];

sigma = 1*  sqrt(Variance_Matrix[min_frame]);
h = 0;
```

```
for(i = 0; i < NumberofLines; i++)
   {
   Distance =
sqrt((Data_Array[min_frame][1]-Data_Array[i][1])*
(Data_Array[min_frame][1]-Data_Array[i][1]) +
(Data_Array[min_frame][2]-Data_Array[i][2])*
(Data_Array[min_frame][2]-Data_Array[i][2]) +
(Data_Array[min_frame][3]-Data_Array[i][3])*
(Data_Array[min_frame][3]-Data_Array[i][3]) +
(Data_Array[min_frame][4]-Data_Array[i][4])*
(Data_Array[min_frame][4]-Data_Array[i][4]) +
(Data_Array[min_frame][5]-Data_Array[i][5])*
(Data_Array[min_frame][5]-Data_Array[i][5]));

   if(Distance < sigma)
      {
      cluster[h] = i;
      h++;
      }
   }

   //Open output file and print first cluster frames with their corresponding principal
components

FILE *fp2;

fp2 = fopen("cluster2.txt", "w");

fseek(fp2, +0, SEEK_SET);

for(i = 0; i < h; i++)
   {
   fprintf(fp2, "%7d %10.7f %10.7f %10.7f %10.7f %10.7f\n",cluster[i]+1,
Data_Array[cluster[i]][1], Data_Array[cluster[i]][2], Data_Array[cluster[i]][3],
Data_Array[cluster[i]][4], Data_Array[cluster[i]][5] );
   }

fclose(fp2);

   //Free allocated memory from matrices

for(i = 0; i < NumberofLines - WINDOW; i++ )
   {
```

```
   free(Data_Array[i]);
   }

free(Data_Array);

free(Variance_Matrix);

exit(0);

}


void *thread( void *input)
{
int fraction = *(int*)input;

printf("Created thread:%d\n", fraction);

int i = 0, h = 0, l = 0, N = 0, start, finish;
float Mean =0.0, minus_sum =0.0, minus_sum_sq =0.0, sum =0.0, sum_sq =0.0, new_sum =0.0,
new_sum_sq = 0.0, val;

   //Calculate the frames that will be calculated in each thread, based on the "fraction" it was
assigned

if(fraction == 1)
   {
   start = 0;
   finish = fraction*mills;
   }

if(fraction < THREADS && fraction != 1)
   {
   start = (mills*(fraction-1))-WINDOW;
   finish = mills*fraction;
   }

if(fraction == THREADS && fraction != 1)
   {
   start = (mills*(fraction-1))-WINDOW;
   finish = NumberofLines;
   }

   //Allocate memory for distance matrix
```

```
float Distance_Matrix[WINDOW][WINDOW];

   //Calculate the distance between frames in the first window
   //Add values to sum and minus sum, in order to help calcualate variance and move to second
window

for(i = start; i < start+WINDOW; i++)
   {
   for(h = i+1; h < start+WINDOW; h++)
      {
      val = Distance_Matrix[i-start][h-i-1] =
sqrt((Data_Array[i][1]-Data_Array[h][1])*
(Data_Array[i][1]-Data_Array[h][1]) +
(Data_Array[i][2]-Data_Array[h][2])*
(Data_Array[i][2]-Data_Array[h][2]) +
(Data_Array[i][3]-Data_Array[h][3])*
(Data_Array[i][3]-Data_Array[h][3]) +
(Data_Array[i][4]-Data_Array[h][4])*
(Data_Array[i][4]-Data_Array[h][4]) +
(Data_Array[i][5]-Data_Array[h][5])*
(Data_Array[i][5]-Data_Array[h][5]));

      sum += val;
      sum_sq += val * val;
      N++;

      if(i == start)
         {
         minus_sum += val;
         minus_sum_sq += val*val;
         }

      }
   }

   //Caslculate variance for the rest of the windows via a single pass algorithm

for(i = start; i < finish - WINDOW; i++)
   {

   //Moves Distance Matrix parameters free memory for next window
```

```
   Mean = sum/N;
   Variance_Matrix[i] = sum_sq / N - Mean * Mean;

   memmove( &Distance_Matrix[0][1], &Distance_Matrix[0][0],
               (WINDOW-1)*WINDOW*sizeof(float) );
   memmove( &Distance_Matrix[0][0], &Distance_Matrix[1][0],
               (WINDOW-1)*WINDOW*sizeof(float) );

   new_sum = 0.0;
   new_sum_sq = 0.0;

   //Calculates distances for new window

   for(h = 1; h < WINDOW; h++)
      {
      val = Distance_Matrix[h-1][0] =
sqrt((Data_Array[h+i][1]-Data_Array[WINDOW+i][1])*
(Data_Array[h+i][1]-Data_Array[WINDOW+i][1]) +
(Data_Array[h+i][2]-Data_Array[WINDOW+i][2])*
(Data_Array[h+i][2]-Data_Array[WINDOW+i][2]) +
(Data_Array[h+i][3]-Data_Array[WINDOW+i][3])*
(Data_Array[h+i][3]-Data_Array[WINDOW+i][3]) +
(Data_Array[h+i][4]-Data_Array[WINDOW+i][4])*
(Data_Array[h+i][4]-Data_Array[WINDOW+i][4]) +
(Data_Array[h+i][5]-Data_Array[WINDOW+i][5])*
(Data_Array[h+i][5]-Data_Array[WINDOW+i][5]));

      new_sum = new_sum + val;
      new_sum_sq = new_sum_sq + val * val;
      }

   //Calculates variance for new window

   sum = sum - minus_sum + new_sum;
   sum_sq = sum_sq - minus_sum_sq + new_sum_sq;

   minus_sum = 0.0;
   minus_sum_sq = 0.0;
   for ( h = 0; h < WINDOW-1; h++ )
      {
      minus_sum += Distance_Matrix[0][h];
      minus_sum_sq += Distance_Matrix[0][h]*Distance_Matrix[0][h];
      }
   }
```

}

6. References

1. Shao, J., Tanner, S. W., Thompson, N., & Cheatham, T. E. (2007). Clustering molecular dynamics trajectories: 1. characterizing the performance of different clustering algorithms. *Journal of Chemical Theory and Computation*, *3*(6), 2312–2334. https://doi.org/10.1021/ct700119m
2. Jamroz, M., & Kolinski, A. (2013). Clusco: Clustering and comparison of protein models. *BMC Bioinformatics*, *14*(1). https://doi.org/10.1186/1471-2105-14-62
3. Lindsey I. Smooth, 2002, A tutorial on Principal Component Analysis
4. Koukos, P. I., & Glykos, N. M. (2014). On the application of good-turing statistics to quantify convergence of biomolecular simulations. *Journal of Chemical Information and Modeling*, *54*(1), 209–217. https://doi.org/10.1021/ci4005817
5. Papaleo, E., Mereghetti, P., Fantucci, P., Grandori, R., & De Gioia, L. (2009). Free-energy landscape, principal component analysis, and structural clustering to identify representative conformations from molecular dynamics simulations: The myoglobin case. *Journal of Molecular Graphics and Modelling*, *27*(8), 889–899. https://doi.org/10.1016/j.jmgm.2009.01.006
6. David, C. C., & Jacobs, D. J. (2013). Principal component analysis: A method for determining the essential dynamics of proteins. *Protein Dynamics*, 193–226. https://doi.org/10.1007/978-1-62703-658-0_11
7. Glykos, N. M. (2006). Software news and updates carma: A molecular dynamics analysis program. *Journal of Computational Chemistry*, *27*(14), 1765–1768. https://doi.org/10.1002/jcc.20482
8. Glykos, N. (2017) Home – plot – Utopia. Available at: https://utopia.duth.gr/glykos/plot/
9. Herbert J. Bernstein. (2009). RasMol. Retrieved from http://www.rasmol.org

10. Joni. (2019, November 16). *Welford's method for computing variance*. The Mindful Programmer. Retrieved February 17, 2023, from https://jonisalonen.com/2013/deriving-welfords-method-for-computing-variance/
11. McKiernan, K. A., Husic, B. E., & Pande, V. S. (2017). Modeling the mechanism of CLN025 beta-hairpin formation. *The Journal of Chemical Physics*, *147*(10), 104107. https://doi.org/10.1063/1.4993207
12. Picone, D., Rivieccio, A., Crescenzi, O., Caliendo, G., Santagada, V., Perissutti, E., Spisani, S., Traniello, S., & Temussi, P. A. (2001). Peptide T revisited: Conformational mimicry of epitopes of anti-HIV proteins. *Journal of Peptide Science*, *7*(4), 197–207. https://doi.org/10.1002/psc.320
13. Wölcke, J., & Weinhold, E. (2001). A DNA-binding peptide from a phage display library. *Nucleosides, Nucleotides and Nucleic Acids*, *20*(4-7), 1239–1241. https://doi.org/10.1081/ncn-100002526
14. Estephan, E., Dao, J., Saab, M.-B., Panayotov, I., Martin, M., Larroque, C., Gergely, C., Cuisinier, F. J. G., & Levallois, B. (2012). SVSVGMKPSPRP: A broad range adhesion peptide. *Biomedizinische Technik/Biomedical Engineering*, *57*(6). https://doi.org/10.1515/bmt-2011-0109
15. Serafeim, A.-P., Salamanos, G., Patapati, K.K. & Glykos*, N.M. (2016), "Sensitivity of Folding Molecular Dynamics Simulations to Even Minor Force Field Changes", J. Chem. Inf. Model., 56, 2035-2041.
16. Gkogka, I. & Glykos*, N.M. (2022), "Folding molecular dynamics simulation of T-peptide, a HIV viral entry inhibitor : Structure, dynamics, and comparison with the experimental data", J. Comput. Chem., 43, 942-952.
17. Alexiadou D. (2022) "Structural computational biology studies of a non-specific DNA-binding peptide", MSc thesis, Dept of Molecular Biology & Genetics, Democritus University of Thrace, Greece.