Recovery of the Trajectory of DNA inside Viral Capsids from Liquid Crystal Vector Fields

Zhijie Wang

December 2020

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Bacteriophages or phages are the most abundant entities on Earth. They are viruses that invade bacterial cells, disrupt metabolism, and cause them to lyse. Phages are composed of proteins that encapsulate a DNA or RNA genome. The capsid is attached to a tail which has fibers, used for attachments to receptors on the bacterial cell surface [6, 15]. Bacteriophages are of great interest for their potential as attractive therapeutic agents against rapidly emerging, antibiotic-resistant bacteria [6]. The applications of phages also range from the diagnosis of diseases, their prevention, and their treatment. Because of their use in the field of biotechnology and medical science, bacteriophages need to be studied [15].

The double-stranded DNA molecule inside the viral capsid is found under extreme concentration and osmotic pressure. This high armatic pressure facilitates this phage genome to enter the host cell. This is a unique feature of bacteriophages since for other viruses, their entire virus structures enter the cell cytoplasm [17]. Because this key step in the expansion process of bacteriophages, the DNA within the bacteriophage capsid must be optimally packed for ejection, with a pressure strong enough to inject the genome into the host cell. Because of the bending energy due to the bending rigidity of the DNA strands compared with the diameter of the capsid, the arrangement of the DNA inside the capsid is ordered in concentric layers near the capsid wall, at the center of the capsid there is a disordered core

that is believed to relieve the high pressure.

At the time of ejecting the DNA molecule, a mechanism suggests a phase transition, possibly into a 'liquid-like' state [8, 14]. Both packing and releasing of the genome are highly dependent on how the DNA folds inside the capsid; however, our understanding of this folding remains limited. There are a few theoretical models attempting to describe the DNA molecules in a liquid crystalline phase [2, 11]. In this thesis, the starting model is an energy minimizing configuration built from cryo-electron microscopy (cryo-EM) data. The model of DNA folding is built with cryo-electron microscopy data, the hexagonal chromonic liquid crystal structure of the packed DNA [10], and the continuum theory of liquid crystals[18]. It also assumes that the hydrated DNA fills the entire volume of the capsid[7].

In this work, we developed three different protocols to recover the trajectory of DNA from liquid crystal vector fields. To facilitate computation, an effective method of sampling vectors from the vector fields is proposed. Also, an optimization method for generating sets of molecules around the directors in liquid crystal vector fields. And finally, we generate the filaments from the sampled vector fields.

This thesis is organized as follows. In Chapter 2, we provide some background on knots and knot invariants. We also present how to compute the dowker code and alexander polynomial of a knot. The basics of optimization are given. We first define the unconstrained and constrained optimization problems, and give the general solutions. Then we introduced an advanced algorithm for solving nonlinear optimization problems with nonlinear equality and inequality constraints. In this work, we will obtain numerical solutions to minimization problem instead of general ones. A basic idea of Monte Carlo sampling method is discussed at the end of background chapter.

The vector field which we are using throughout this work is described at the beginning of Chapter 3. For every vector in the vector field, it is the liquid crystal director, which is the mean of surrounding organic molecules. To obtain the liquid crystal molecules, we formulate the optimization problem of generating microvectors (molecules) around a macrovector

(director). Then, because of the large dimension of the dataset, we consider a way to sample uniformly from the vector field. Following this, we develop three different protocols for generating filaments from the vectors sampled from the vector field. This connecting curve recovers the trajectory of the DNA inside viral capsids. At the end of this chapter, we compute the alexander polynomials of the generated trajectories.

In Chapter 4, we discuss the results of the optimization problem of generating microvectors, with respect to different parameters, such as: numbers of microvectors, maximal angles between microvectors and the macrovectors, and how initial values affect the performance of this algorithm. We then present some figures of the sampling method. From this, we connect the sampled vector fields using three available protocols. The advantages and limitations of them are discussed as well. From the trajectories, we illustrate the results of Alexander polynomials and findings of other knot properties. In Chapter 5, the thesis closes with concluding remarks and possible directions for future work.

Chapter 2

Background

2.1 Knot Theory

Definition 2.1.1 ([9])**.** A link L of m components is a subset of $\mathbb{R}^3$, that consists of m disjoint, piecewise linear, simple closed curves. A link of one component is a *knot*. See Figure 2.1a and 2.1b.

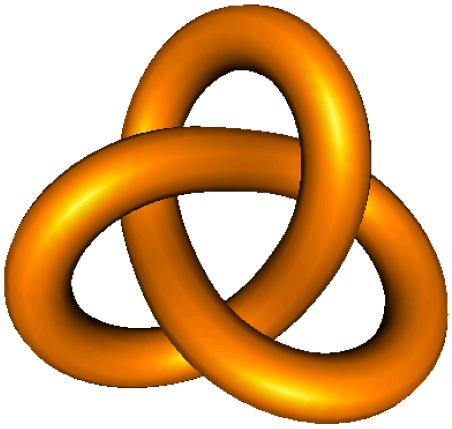
Definition 2.1.2 ([9])**.** Links L_1 and L_2 in $\mathbb{R}^3$ are equivalent if there is an orientation-preserving picewise linear homeomorphism $h : \mathbb{R}^3 \to \mathbb{R}^3$ such that $h(L_1) = (L_2)$.

Definition 2.1.3 ([5])**.** Equivalent knots are said to be of the same *type*, and each equivalence class of knots is a *knot type*.

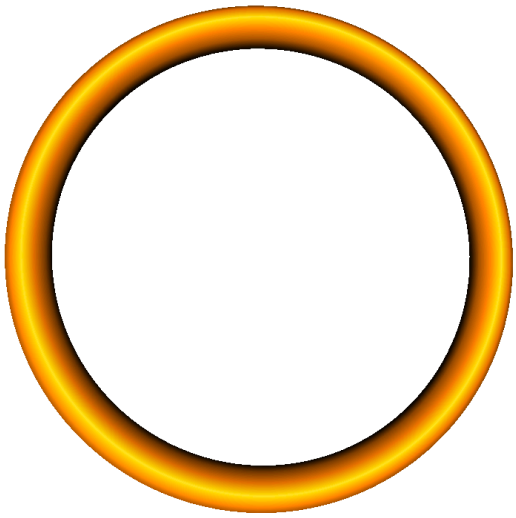
Definition 2.1.4 ([9])**.** A knot K is said to be the *unknot* if it bounds an embedded piecewise linear disk in $\mathbb{R}^3$.

Definition 2.1.5 ([9])**.** A knot K is a *prime knot* if it is not the unknot, and if $K \equiv K_1 + K_2$ implies that K_1 or K_2 is the unknot.

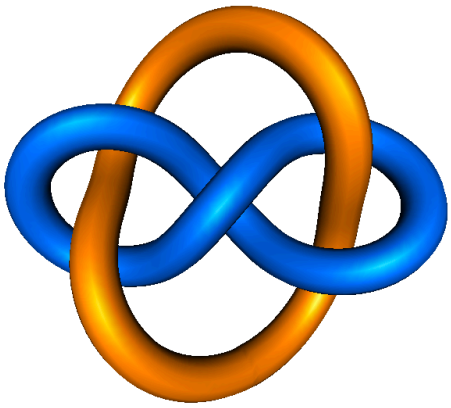
Definition 2.1.6 ([9])**.** The image of a link L in $\mathbb{R}^2$ together with "over and under" information at the crossings is called a *link diagram* of L



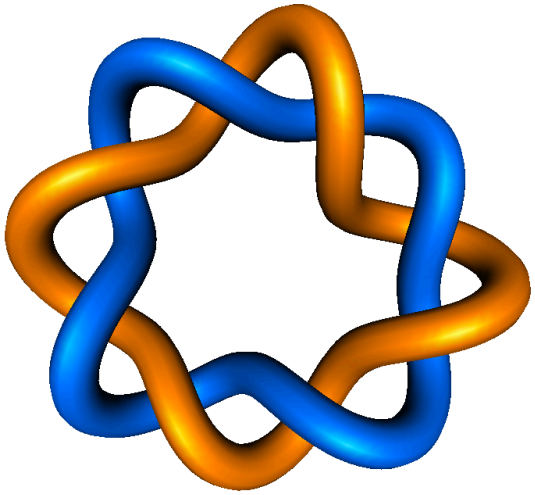
(a) Trefoil knot 3_1, prime knot



(b) Knot 0_1, also called *unknot*



(c) Link 5_1^2



(d) Link 8_1^2

Figure 2.1: Common Knots and Links

We call a picture of a knot a projection of the knot. Figure 2.1a is a projection of the trefoil knot and figure 2.1b is a projection of the unknot.

A crossing in a diagram of an oriented link can be allocated a sign; the crossing is said to be positive or negative, or to have sign $+1$ or -1. The standard convention is shown in Figure 2.2.

Definition 2.1.7 ([9])**.** The *Reidemeister moves* are of three types, shown in Figure 2.3;

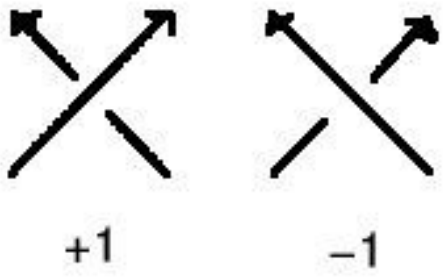


Figure 2.2: +1 and -1 crossings

each replaces a simple configuration of arcs and crossings by another configuration.

I Twist and untwist in either direction.

II Move one loop completely over another.

III Move a string completely over or under a crossing.

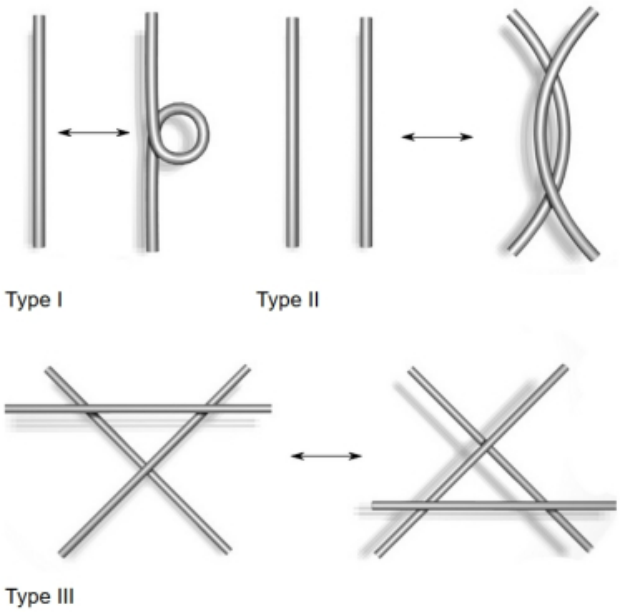


Figure 2.3: Three types of Reidemeister moves

The Dowker notation is a simple way to describe a diagram of a knot. Suppose we want to describe a 5_1 knot, like the one in Figure 2.4. We pick a starting point and an orientation

as in Figure 2.5. When we reach the first + crossing, we label it +1. Leaving that crossing along the understrand in the direction of the orientation, label the next crossing that you come to with a 2. Continue labeling that crossing on the same strand of the knot, and label the next crossing with a 3. Keep labeling the crossings with integer sequence until you go through all the way around the knot once. When done, each crossing in the knot will have two labels on it because every crossing is passed through twice. And each crossing will have one even number and one odd number labeling it [1].

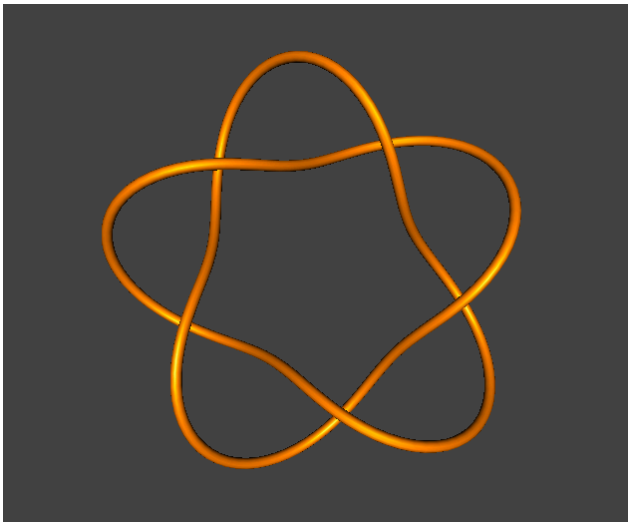


Figure 2.4: A 5_1 knot

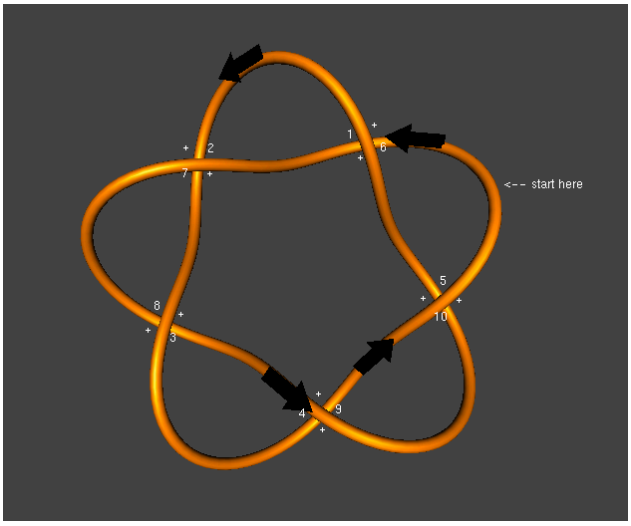


Figure 2.5: Label each crossing of the knot with two numbers.

2.1.1 Alexander Polynomial

The idea of knot polynomial is to find a way to tell knots apart. For each knot, we can associate a polynomial with it. We compute the polynomial from a projection of the knot, but different projections of the same knot would yield the same polynomial. Therefore, we say the knot polynomial is an invariant of the knot [1].

The Alexander polynomial is the first polynomial discovered by J. Alexander in about 1928. It is good at distinguishing knots and links.

Below I will describe the procedure to obtain the Alexander polynomial of a knot.

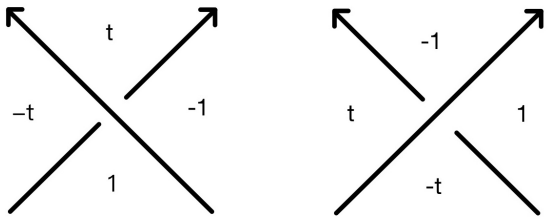


Figure 2.6: Notation of crossings used in Alexander polynomial

1. Take an oriented diagram D for a knot K and number the crossing $1, ..., n$, the regions $1, ..., n+2$.

2. Create an $n \times (n+2)$ matrix M with $M_{i,j} = 0$ if region j doesn't touch crossing i, otherwise see Figure 2.6.

   - on the left before undercrossing: $-t$
   - on the right before undercrossing: 1
   - on the left after undercrossing: t
   - on the right after undercrossing: -1

3. New matrix: $\widetilde{M} = M$ with any two adjacent columns deleted.

4. $\Delta_K(t) = det(\widetilde{M})$

To understand the computation process more easily, we let K be the trefoil knot. Observe from Table 2.1, we can delete the first two column and obtain matrix $\widetilde{M} = \begin{pmatrix} 0 & -1 & t \\ -t & 0 & t \\ -1 & -t & t \end{pmatrix}$.

Then we have $\Delta_K(t) = det(\widetilde{M}) = t^3 - t^2 + t$ which is the Alexander polynomial of knot K.

	r1	r2	r3	r4	r5
c1	1	-t	0	-1	t
c2	1	-1	-t	0	t
c3	1	0	-1	-t	t

Table 2.1

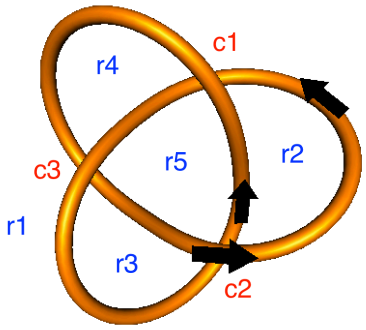


Figure 2.7

2.2 Optimization

For the purpose of introducing optimization problem, I will use *A First Course in Numerical Methods* [3].

There are several types of optimization problems. The prototype used here is the minimization of a scalar function ϕ in n variables $\mathbf{x} = (x_1, x_2, ..., x_n)^T$. We write this as

$$\min_{x \in \mathbb{R}^n} \phi(\mathbf{x})$$

which require that $\mathbf{x}$ be in $\mathbb{R}^n$ or a subset of it that is characterized by one or more constraints.

2.2.1 Unconstrained Optimization

Here $\phi : \mathbb{R}^n \to \mathbb{R}$. This means that $\mathbf{x} = (x_1, x_2, ..., x_n)^T$ is a vector, but ϕ takes on scalar values.

Note in the following, $\mathbf{x}^*$ is the local minimizer, and $\phi(\mathbf{x}^*)$ is the local minimum. The *minimum* is the minimal value of a function, while the point attains the minimum is called *minimizer*.

Theorem 2.2.1 (Unconstrained Minimization Conditions)**.** Assume that $\phi(\mathbf{x})$ is smooth enough, e.g., suppose it has all derivatives up to third order, which are continuous and bounded. Then:

- A *necessary condition* for having a local minimum at a point $\mathbf{x}^*$ is that $\mathbf{x}^*$ be a critical point, i.e.,

  $$\nabla\phi(\mathbf{x}^*) = 0,$$

  and that the symmetric Hessian matrix $\nabla^2\phi(\mathbf{x}^*)$ be positive semidefinite.

- A *sufficient condition* for having a local minimum at a point $\mathbf{x}^*$ be a critical point and that $\nabla^2\phi(\mathbf{x}^*)$ be a positive definite.

Generally speaking, the condition for a critical point yields a system of nonlinear equations

$$\mathbf{f}(\mathbf{x}) \equiv \nabla\phi(\mathbf{x}) = 0$$

that we need to solve numerically.

Algorithm 2.2.2 (Newton's Method for Unconstrained Minimization)**.** Consider the problem of minimizing $\phi(\mathbf{x})$ over $\mathbb{R}^n$, and let $\mathbf{x}_0$ be a given initial guess.
for k = 0,1,..., until convergence

$$\text{solve } \nabla^2\phi(\mathbf{x}_k)\mathbf{p}_k = -\nabla\phi(\mathbf{x}_k) \text{ for } \mathbf{p}_k$$

$$\text{set } \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{p}_k$$

end

2.2.2 Constrained Optimization

Constrained optimization problem is still optimizing the same scalar function in n variables, $f(\mathbf{x})$ as in unconstrained optimization problems. The difference can be inferred by the name that now there are *equality* and *inequality constrains* that any eligible $\mathbf{x}$ must satisfy. In fact, such constrained optimization problems are of great interest in real life applications. It is important in fields of space technology, robotics, movement sequences in sports, and the control of chemical processes and power plants, to name just a few.

The general problem can be set up as

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \leq 0, i = 1, ..., m \\ & h_j(\mathbf{x}) = 0, j = 1, ..., p \end{aligned} \tag{2.1}$$

with variable $\mathbf{x} \in \mathbb{R}^n$. We also assume its domain $\mathcal{D} = \bigcap_{i=0}^{m} \mathbf{dom}\ g_i \ \cap \ \bigcap_{i=1}^{p} \mathbf{dom}\ h_i$ is nonempty, and denote the optimal value of problem 2.1 by p^*. This means that a point $x \in \mathcal{D}$ is *feasible* if it satisfies the constrains $g_i(x) \leq 0, i = 1, ..., m$ and $h_j(x) = 0, j = 1, ..., p$. Also, the problem is said to be feasible if there exists at least one feasible point, and *infeasible* otherwise. The set of all feasible points is called the *feasible set* $\mathbf{x}$. We say $\mathbf{x}^*$ is the *optimal solutions* or solves the optimization problem if $\mathbf{x}^*$ is feasible and $f(\mathbf{x}^*) = p^*$.

A *convex optimization problem* is one of the form

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \leq 0, i = 1, ..., m \\ & a_i^T \mathbf{x} = b_i,\ j = 1, ..., p \end{aligned} \tag{2.2}$$

where f and $g_1, ...g_m$ are convex functions. The main differences between it and the general

constrained optimization problem 2.1 is that

- the objective function must be convex,
- the inequality constraint functions must be convex,
- the equality constraint functions $h_j(\mathbf{x}) = a_i^T\mathbf{x} - b_i$ must be affine.

A fundamental property of convex optimization problems is that any locally optimal point is also (globally) optimal. This is the reason why convex functions are preferable when setting up the problem.

The Lagrangian and Karush-Kuhn-Tucker Conditions

The basic idea in Lagrangian duality is to take the constrains into account by augmenting the objective function with a weighted sum of the constraint functions [4]. We define the *Lagrangian* $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ associated with the problem 2.1 as

$$L(\mathbf{x}, \lambda, \mu) = f(\mathbf{x}) + \sum_{i=1}^{m} \mu_i g_i(\mathbf{x}) + \sum_{j=1}^{p} \lambda_j h_j(\mathbf{x}) \tag{2.3}$$

where μ_i and λ_j are called the *Lagrangian multipliers* associated with the ith inequality constraint $g_i(\mathbf{x}) \leq 0$ and jth equality constraint $h_j(\mathbf{x}) = 0$ respectively. The vectors μ and λ are the *dual variables* or *Lagrange multiplier vectors* associated with problem 2.1.

Then, we discuss the first-order necessary conditions for $\mathbf{x}^*$ to be a local minimizer in constrained optimization. We are referring to the *Lagrangian function* in 2.3 for our discussion below.

The necessary conditions defined in the following are called *first-order conditions*, also often known as *Karush-Kuhn-Tucker conditions* or *KKT conditions* for short [13].

1. Stationarity Condition

$$0 \ \in \ \partial \left(f(\mathbf{x}^*) + \sum_{i=1}^{m} \mu_i g_i(\mathbf{x}^*) + \sum_{j=1}^{p} \lambda_j h_j(\mathbf{x}^*) \right) \tag{2.4}$$

2. Primal feasibility

$$\begin{aligned} g_i(\mathbf{x}^*) &\leq 0, \ i = 1, ..., m \\ h_j(\mathbf{x}^*) &= 0, \ j = 1, ..., p \end{aligned} \tag{2.5}$$

3. Dual feasibility

$$\mu_i \geq 0, \ i = 1, ..., m \tag{2.6}$$

4. Complementary Slackness

$$\mu_i g_i(\mathbf{x}^*) = 0, \ i = 1, ..., m \tag{2.7}$$

The condition in Eq. 2.7 is implying that either constraint i is active or $\mu_i = 0$, or possibly both. Note that the Lagrange multipliers corresponding to inactive constraints are zero.

If the problem is convex, the KKT conditions are sufficient for the points to be optimal. To see this, note that the second condition shows that $\mathbf{x}^*$ is a feasible solution for problem 2.2. Since $\mu_i \geq 0$, $L(\mathbf{x}^*, \lambda, \mu)$ is convex in $\mathbf{x}^*$; the first condition states that its gradient with respect to $\mathbf{x}$ vanishes at $\mathbf{x}^*$, so it follows that $\mathbf{x}^*$ minimizes $L(\mathbf{x}, \lambda, \mu)$ over $\mathbf{x}$.

When the problem is of small dimensions and is linear, solving the equations from KKT conditions stated from 2.4 to 2.7 gives us analytical solutions. However, to solve a large-scale system of non-linear equations, the Newton's Method is powerful and a different interpretation of KKT conditions is needed.

Another way to approximately formulate the inequality constrained problem as a convex equality constrained problem is the *barrier method*:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}) + \textstyle\sum_{i=0}^{m} I\_(g_i(\mathbf{x})) \\ \text{subject to} \quad & h_j(\mathbf{x}) = 0, \ j = 1, ..., p \end{aligned} \tag{2.8}$$

where $I\_ : \mathbb{R} \to \mathbb{R}$ is the indicator function for nonpositive reals.

Theorem 2.2.3 (Indicator function of a convex set [4])**.** Let $C \subseteq \mathbb{R}^n$ be a convex set, and consider the convex function I_C with domain C and $I_C(x) = 0$ for all $x \in C$. Then we call I_C the *indicator function* of C. In other words, the function is identically zero on the set C.

The idea in the barrier method is to approximate the indicator function $I\_$ by the function

$$I\_(u) = -(1/t)\log(-u), \quad \textbf{dom } I\_ = -\mathbb{R}_{++}, \tag{2.9}$$

where $t > 0$ is a parameter that sets the accuracy of the approximation. The objective function here is convex, because $-(1/t)\log(-u)$ is convex and increasing in u, and also differentiable.

To solve this approximate problem, an algorithm is proposed in [16] and uses one of two main types of steps at each iteration:

- A Newton step attempting to solve the KKT equations 2.4 and 2.7 for the approximate problem via a linear approximation.
- A conjugate gradient (CG) step, using a trust region.

Here we will re-write the optimization problem in a more generalized form. The key idea is the same as discussed above.

$$\begin{aligned} \text{minimize} \quad & f_\mu = f(x) - \mu \sum_i \ln(s_i) \\ \text{subject to} \quad & g(x) + s = 0 \\ & h(x) = 0 \end{aligned} \tag{2.10}$$

The algorithm first attempts to take a Newton step. If it cannot, it attempts a CG step. At each iteration the algorithm decreases with a *merit function*, such as

$$f_\mu(x, s) + v\|h(x), g(x) + s\|$$

to force the solution towards feasibility. A *merit function* is a modified cost function to circumvent poor convergence behavior in iteration regimes where constraints should be weakened.

A brief overview of the Newton step and the CG step used in the algorithm will be given below. Details will not be included in this thesis.

Newton Step

The Newton step is solving the equation 2.4 and 2.7 from KKT conditions making use of an LDL factorization of the matrix. If the fatorization shows that the Hessian is not positive definite, the algorithm uses a conjugate gradient step.

$$\begin{bmatrix} H & 0 & J_h^T & J_g^T \\ 0 & S\Lambda & 0 & -S \\ J_h & 0 & I & 0 \\ J_g & -S & 0 & I \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta s \\ -\Delta y \\ -\Delta \lambda \end{bmatrix} = \begin{bmatrix} \nabla f - J_h^T y - J_g^T \lambda \\ S\lambda - \mu e \\ h \\ g + s \end{bmatrix} \tag{2.11}$$

with variables

- H denotes the Hessian (second derivatives matrix) of the Lagrangian of f_μ in equation 2.10:
$$H = \nabla^2 f(x) + \sum_i \mu_i \nabla^2 g_i(x) + \sum_j \mu_j \nabla^2 h_j(x). \tag{2.12}$$
- J_g denotes the Jacobian (derivatives matrix) of the constraint function g.
- J_h denotes the Jacobian of the constraint function h.
- $S = \text{diag}(s)$.
- λ denotes the Lagrange multiplier vector associated with constraints g.
- $\Lambda = \text{diag}(\lambda)$.
- y denotes the Lagrange multiplier vector associated with h.

- e denote the vector of ones the same size as g.

Conjugate Gradient Step

This method is to minimize a quadratic approximation to the approximate problem in 2.10 in a trust region, subject to linearized constraints.

Let R denotes the radius of the trust region, and other variables defined in the same way in Newton step.

The algorithm first approximately solve the KKT equation 2.4. Then it takes a step $(\Delta x, \Delta s)$ to approximately solve another minimization problem:

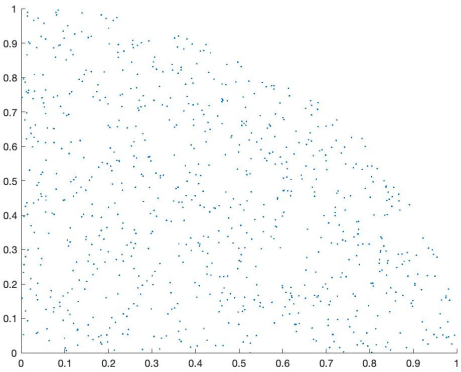
$$\begin{aligned} \text{minimize} \quad & \nabla f^T \Delta x + \tfrac{1}{2}\Delta x^T \nabla^2_{xx} L \Delta x + \mu e^T S^{-1} \Delta s + \tfrac{1}{2}\Delta s^T S^{-1} \Lambda \Delta s \\ \text{subject to} \quad & g(x) + J_g \Delta x + \Delta s = 0 \\ & h(x) + J_h \Delta x = 0 \end{aligned} \tag{2.13}$$

2.3 Monte Carlo

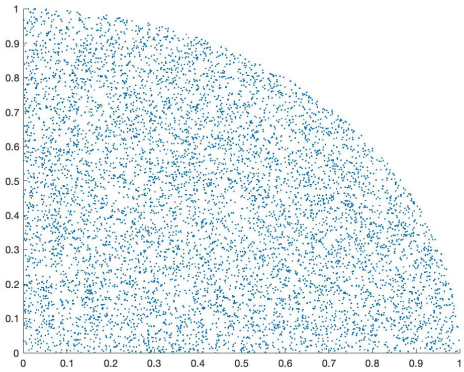
Monte Carlo (MC) methods are a subset of computational algorithms that use the process of repeated random sampling to make numerical estimations of unknown parameters. They are widely used in the fields of physics, game theory, and finance and lead to groundbreaking discoveries. For example, we learn from primary school geometry classes that π is an irrational number, meaning it has infinite digits without any pattern. Now, with Monte Carlo methods, we are able to estimate π to as many digits as we like. By generating random two-dimensional points within a box and counting the number of points which falls in an embedded circle, we can estimate π to any degree of accuracy.

It can be seen from Figure 2.8 that the approximation approaches the exact value of π as we increase the number of points used in the simulation process. And it is reasonably expected that as the number of points $\to \infty$, the estimation $\to$ 3.1415926...

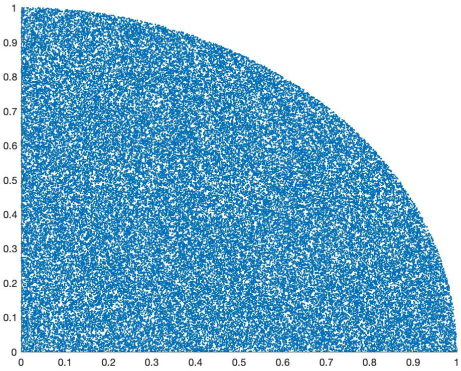
During a Monte Carlo simulation, values are sampled at random from the input probability distributions. Each set of samples is called an iteration, and the resulting outcome



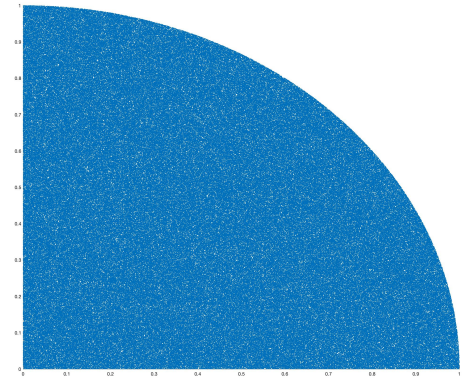
(a) Estimate of π with 1,000 points: 3.092



(b) Estimate of π with 10,000 points: 3.1208



(c) Estimate of π with 100,000 points: 3.14724



(d) Estimate of π with 1 million points: 3.1416

Figure 2.8: Estimation of π using Monte Carlo method

from that sample is recorded. Monte Carlo simulations sample points hundreds or thousands of times, and the result is a probability distribution of possible outcomes. In our example before, the estimated values are just from one iteration. If we perform more simulations on each number of points selected, we are likely to get more accurate values than the current ones listed in the figure.

It is important here that the locations of our randomly generated points must be uniformly distributed throughout the area. If not, the approximation result will be poor. It is implying that by using different probability distributions, variables can have different probabilities of different outcomes occurring.

Chapter 3

Data and Methods

3.1 Vector Field Produced by PDEs

Input data were the solutions to the PDEs described in [19]. The volume (capsid) has been decomposed into around 200,000 tetrahedra as shown in Figure 3.1. With Finite Element method, the solutions to the PDEs are a vector field in which the origin of each vector is placed at one of the vertices of a tetrahedron in which the domain (volume encapsulated by the viral capsid) has been decomposed for integration. In Figure 3.1, each tetrahedron was given by four three dimensional coordinates corresponding to the positions of its vertices. On each vertex, there is a vector N relating to it. Figure 3.2 shows the vector field indicating the value of the order parameter (degree of orientation) s by the color of the vector (small degree is depicted in red and high degree in blue). Degree of orientation illustrates how well-ordered the DNA strands are, e.g. s=0 indicates that the DNA strands are oriented in all direction equally[19]. For full description and samples of data files, please refer to Appendix.

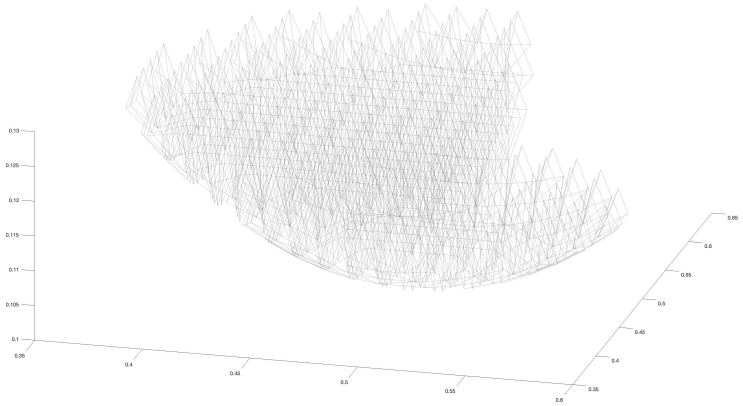


Figure 3.1: Triangulation of the volume (capsid)

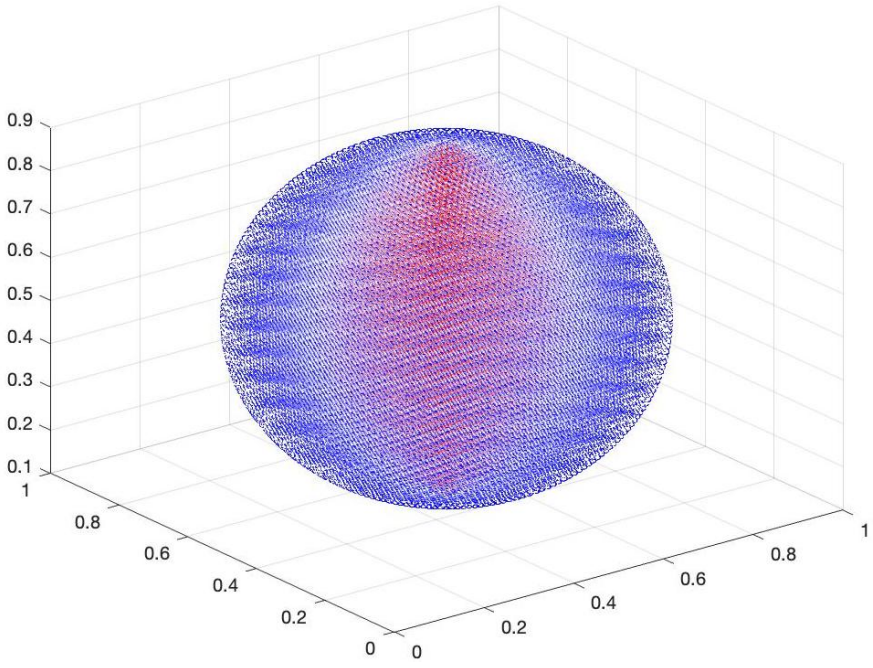


Figure 3.2: The complete vector field

3.2 Use KKT to Find Microvectors

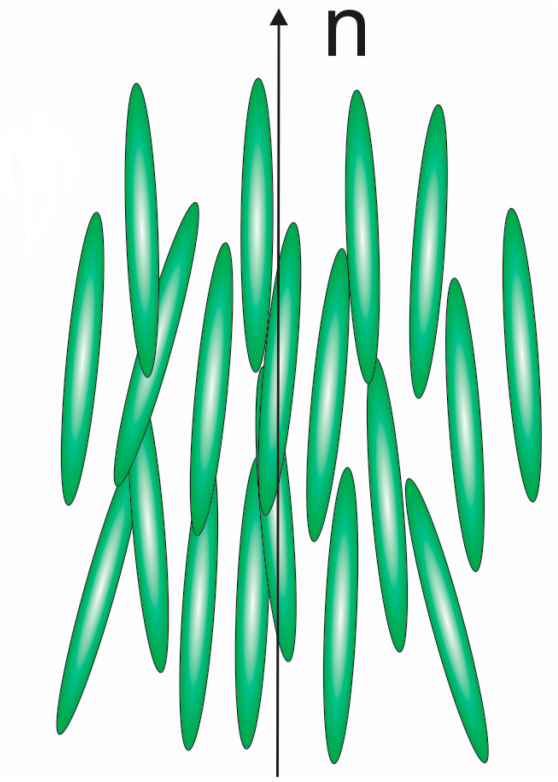


Figure 3.3: Liquid crystal director and microvectors

The background behind this problem is that in a nematic liquid crystal, rod-like organic molecules are on average spontaneously oriented along the direction called the director $\mathbf{n}$ as shown in Figure 3.3. Now we obtain the director, the macrovector, and we want to reconstruct the organic molecules, microvectors, around $\mathbf{n}$ [12].

In this section, our problem is to generate a set of vectors $\mathbf{u_i}$ for $i = 1, ..., m$ such that the mean of these so-called "microvectors" is the macrovector $\mathbf{n}$.

We can interpret this problem as given $\mathbf{n}$, find $\mathbf{u_i}$ such that

$$\frac{1}{m}\sum_{i=1}^{m}\mathbf{u_i} = \mathbf{n},$$

and for each randomly generated microvector, the euclidean norm of it, $\mathbf{u_i}$, should be 1

$$\|\mathbf{u_i}\| = 1$$

Also, we denote the angle between the macrovector and each microvector as θ_i, and it should be less than $\frac{\pi}{2}$. For this problem we can use

$$cos(\theta_i) = \frac{\mathbf{u_i n}}{\|\mathbf{u_i}\|\|\mathbf{n}\|}$$

To translate our problem to an optimization problem, first let $\mathbf{u_i} = (x_i, y_i, z_i)$ and $\mathbf{n} = (a, b, c)$. Then our intuition leads the following formulation of problem:

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{i=1}^{m} \|\mathbf{u}_i - \mathbf{n}\| \\ \text{subject to} \quad & cos(\theta_i) \geq p, \quad i = 1, ..., m \\ & \|\mathbf{u}_i\| = 1, \qquad i = 1, ..., m \end{aligned} \tag{3.1}$$

Here p determines the maximal angles between the microvectors $\mathbf{u_i}$ and the macrovector $\mathbf{n}$.

We can first try to see how this algorithm looks like when generating two microvectors.

Now we re-formulate this optimization problem to be more computer readable:

$$\begin{aligned} \text{minimize} \quad & \sqrt{(x_1-a)^2+(y_1-b)^2+(z_1-c)^2} + \sqrt{(x_2-a)^2+(y_2-b)^2+(z_2-c)^2} \\ \text{subject to} \quad & \tfrac{x_1a+y_1b+z_1c}{\sqrt{x_1^2+y_1^2+z_1^2}\sqrt{a^2+b^2+c^2}} \geq p \\ & \tfrac{x_2a+y_2b+z_2c}{\sqrt{x_2^2+y_2^2+z_2^2}\sqrt{a^2+b^2+c^2}} \geq p \\ & \sqrt{x_1^2+y_1^2+z_1^2} = 1 \\ & \sqrt{x_2^2+y_2^2+z_2^2} = 1 \end{aligned} \tag{3.2}$$

However, the objectives in Functions 3.1 and 3.2 are not convex, neither those constraint equalities and inequalities. And it is unrealistic to list all equations by hand if the goal is to generate more than 100 microvectors. Therefore, we can modify the optimization problem again and get the vector form of it.

$$\begin{aligned} \text{minimize} \quad & e^T d \\ \text{subject to} \quad & p - \frac{(\mathbf{x}\cdot a + \mathbf{y}\cdot b + \mathbf{z}\cdot c)^2}{\mathbf{x}^2+\mathbf{y}^2+\mathbf{z}^2} \leq 0 \\ & \mathbf{x}^2 + \mathbf{y}^2 + \mathbf{z}^2 - e = 0 \end{aligned} \tag{3.3}$$

where $d = (\mathbf{x} - a)^2 + (\mathbf{y} - b)^2 + (\mathbf{z} - c)^2 \in \mathbb{R}^{m\times 1}$, and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^{m\times 1}$ are vectors of x, y, z-coordinates of microvectors u respectively. We are introducing e, a column vector whose entries are all 1's, because we want to sum the Euclidean distances between all microvectors and macrovector and give the value in a linear form. And the reason why we eliminate the term $\sqrt{a^2 + b^2 + c^2}$ in the denominators of the fractions in inequality constraints is we observed this value equals 1 for all macrovectors we want to study ($\|\mathbf{n}\| = 1$ for all $\mathbf{n}$).

Now the system of equations are formulated appropriately for computation. To get the solutions, solving the KKT conditions is sufficient because both objective and all constraint functions are convex.

But, as mentioned in the background section, a variation of KKT conditions is needed for our case because of the large dimensions of $d, \mathbf{x}, \mathbf{y}, \mathbf{z}$. The method described in 2.10 is used because it handles the nonlinear objective and constraint functions well.

3.3 Randomly Select Subset of Microvectors

Given that we have around 200,000 vectors in the vector field, the computation resources required for connecting this number of vectors and computing the knot type of the connected curve are too high. There is a crucial need for appropriate sampling. In this section, the methods of sampling vectors are described so that the we are able to use the subsam-

ple to connect the vectors and obtain different possibilities of dsDNA filaments inside the bacteriophage capsid.

There are multiple methods to do the sampling from a large data set.

Because the shape of the vector field is spherical, it is undesirable to simply select spherical coordinates, the longitude θ and the colatitude ϕ from a uniform distribution. In such a distribution, $\theta \in [0, 2\pi)$ and $\phi \in [0, \pi]$, the mapping from spherical to Cartesian coordinates does not preserve the area − the spherical space is pinched and compressed at the poles by the mapping. It can be easily illustrated by creating and looking at the figures of such a distribution would bring us. In Figure 3.4, 5,000 points are plotted to see the uniform distribution of ϕ and θ. Then we map these points to a sphere from spherical coordinates into the Cartesian space. The mapping functions are

$$x = r \sin\phi \cos\theta$$

$$y = r \sin\phi \sin\theta$$

$$z = r \cos\phi.$$

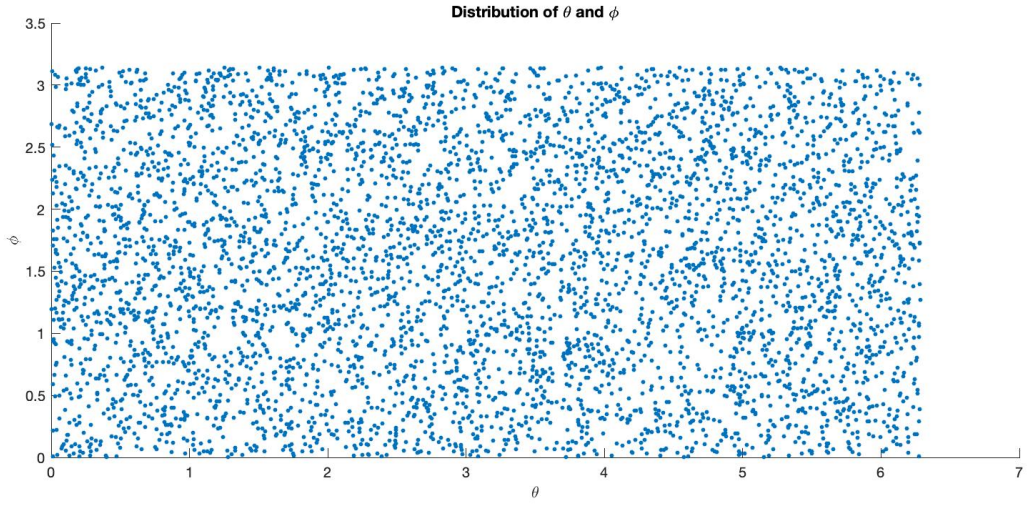


Figure 3.4: Uniform distribution of ϕ and θ

Then we obtain a set of points in the Cartesian space. From the visualizations in Fig-

ure 3.5, we see that the points are clearly not uniformly distributed but dense in north and south poles.

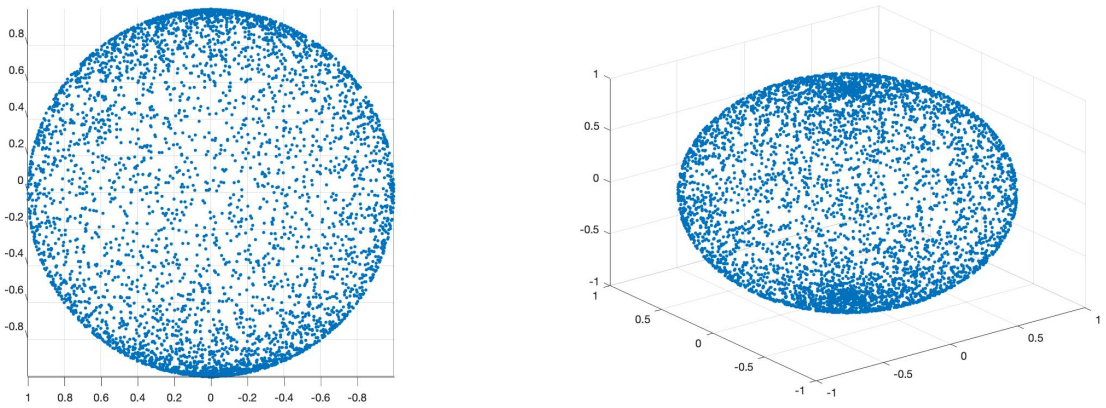


Figure 3.5: Cartesian space views of "uniformly" distributed points

In order to fix this, a different sampling method is needed.

The idea is that we first divide the volume into multiple spherical shells. In each shell, we count the portion of vectors in the original vector field, and we use this portion to determine the number of vectors in the newly sampled vector field. Then we randomly generate vectors inside the shell and include the closest vector in the original vector field to it in our sample.

The details of the modified algorithm are as follows:

1. Convert all vectors to spherical coordinates, and obtain the radius of the spherical vector field r. The radius is

$$r = \max_{\forall v \in V} \|v - c\|,$$

where c is the center of the vector field, and V denotes the whole set of vectors in the field.

2. Divide the spheres into p spherical shell. The $i-$th shell is the region of a ball between two concentric spheres of radii $r_i = \frac{i \cdot r}{p}$ and r_{i-1}.

3. For ith shell, $i = 1, ..., p$, count the number of vectors n_i inside this shell. Let $m_i = \frac{n_i}{n}$

be the proportion of vectors for the ith shell, where n denotes the total number of vectors in V.

4. Let k be the number to sample from V, where $0 \leq k \leq n$. Inside the i th shell, there will be $k_i = k \cdot m_i$ vectors for the subsample.

5. Set $\phi = \sin^{-1}(\gamma)$ where γ is a random value drawn from the standard uniform distribution on the open interval $(-1, 1)$. θ is also a random value drawn from uniform distribution but on the open interval $(0, 2\pi)$. r is randomly drawn from uniform distribution on open interval (r_{i-1}, r_i).

6. Notice the vector $u = (\theta, \phi, r)$ only represents a random vector inside the sphere but not necessarily an element of V. This step determines the vector $s \in V$ with smallest Euclidean distance with u. In other words, we want s such that

$$\|s - u\| = \min_{v \in V} \|v - u\|.$$

   Then we include this vector s in the subsample.

7. Repeat Step 5 and 6 until we obtain k_i vectors for current shell i. Then we go to the next shell $i + 1$ for the same sampling process.

3.4 Generate Closed Curves

There are different approaches to generate an ensemble of curves from the set of vectors describing the microstructure of the liquid crystal.

To facilitate calculations, cylindrical coordinates are used. Let $v_{i,head}$ be the position of current vector's head, $v_{i+1,tail}$ be the position of next vector's tail, and $x_{i,head}$ be the x-coordinate of current vector's head. Similar notation will be used for y, z, θ. Here, we have $v_{i,head} = (x_{i,head}, y_{i,head}, z_{i,head})$. V denote the whole vector field.

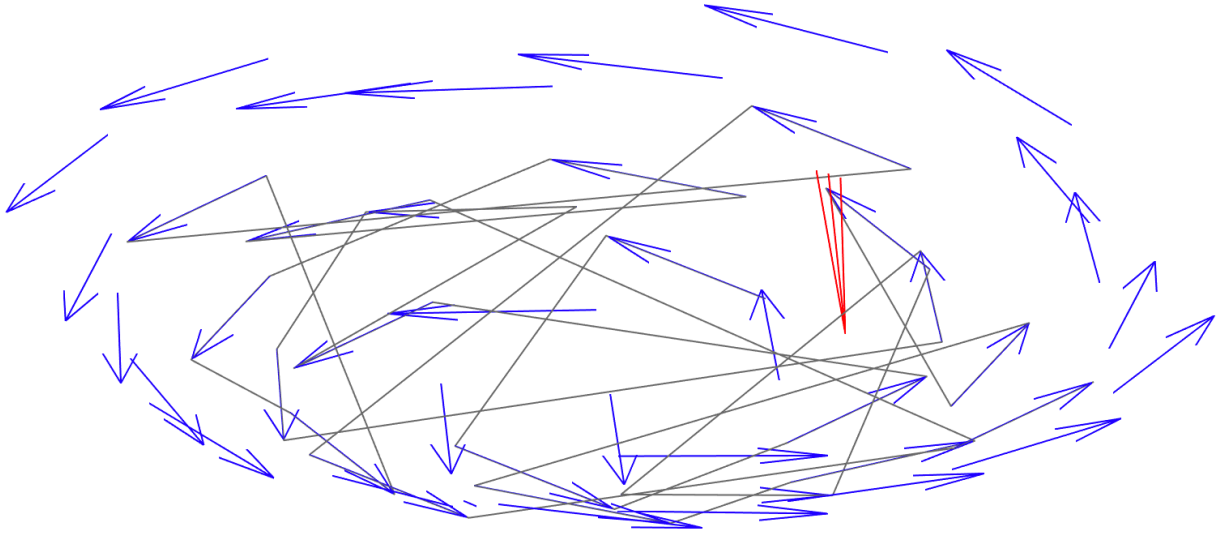


Figure 3.6: CP1 connects the first 20 vectors

The simplified **CP 1** is connecting vectors as follows:

1. Convert all coordinates to cylindrical coordinates.
2. Group vectors by the z_{tail} into layers.
3. Randomly pick a vector from the lowest layer (smallest z_{tail}).
4. Connect all vectors in the same layer according to their azimuth angles (θ) from the smallest to the largest.
5. Go to the next layer and repeat last step until all layers are discovered.
6. Close the knot by connecting the head of last vector and the tail of first vector.

The simplified **CP2** is as follows (complete algorithm is provided in the Appendix):

1. Convert all coordinates to cylindrical coordinates.
2. Group vectors by both z_{head} and z_{tail}.

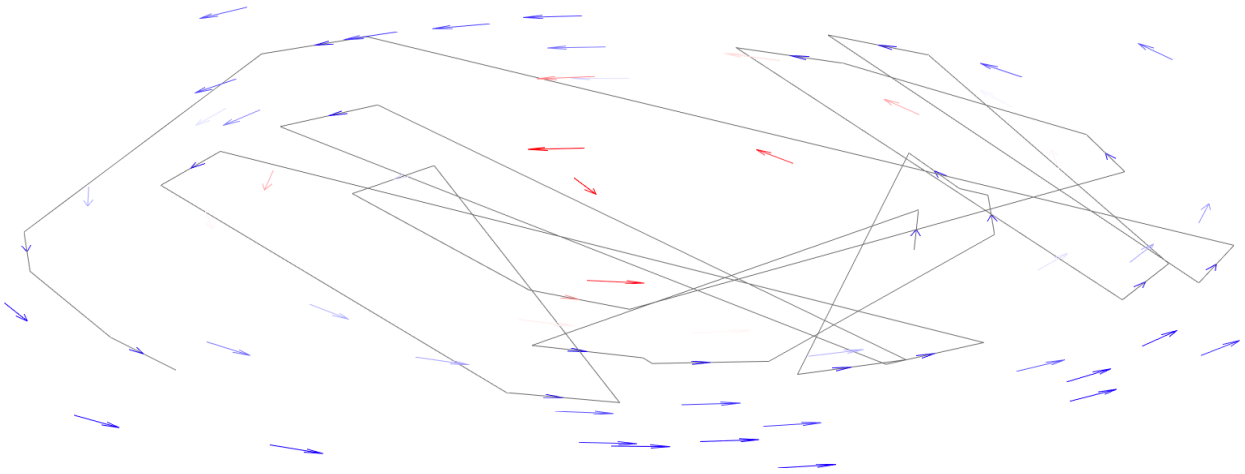


Figure 3.7: CP2 connects the first 20 vectors from a 20,000 vectors sample

3. Randomly pick a vector from V.

4. Find the next vector whose tail has the same height as the previous vector's head, and closest θ. Then connect these two vectors.

5. Go to find the next vector until there is no vector's tail having the height as current vector's head.

6. Close the knot by connecting the head of last vector and the tail of first vector.

The simplified **CP3** is as follows (complete algorithm is provided in the Appendix):

1. Convert all coordinates to cylindrical coordinates.

2. Group vectors by the z-coordinate of both their heads and tails.

3. Randomly pick a vector from the bottom of the vector field (lowest layer with smallest z_tail).

4. Find the next vector using the criterion from Protocol 2, and also restricting the distance between next vector's tail $v_{i+1,tail}$ and current vector's head $v_{i,head}$ with a self-tuning tolerance tol such that $\|v_{i+1,tail} - v_{i,head}\| \leq tol$.

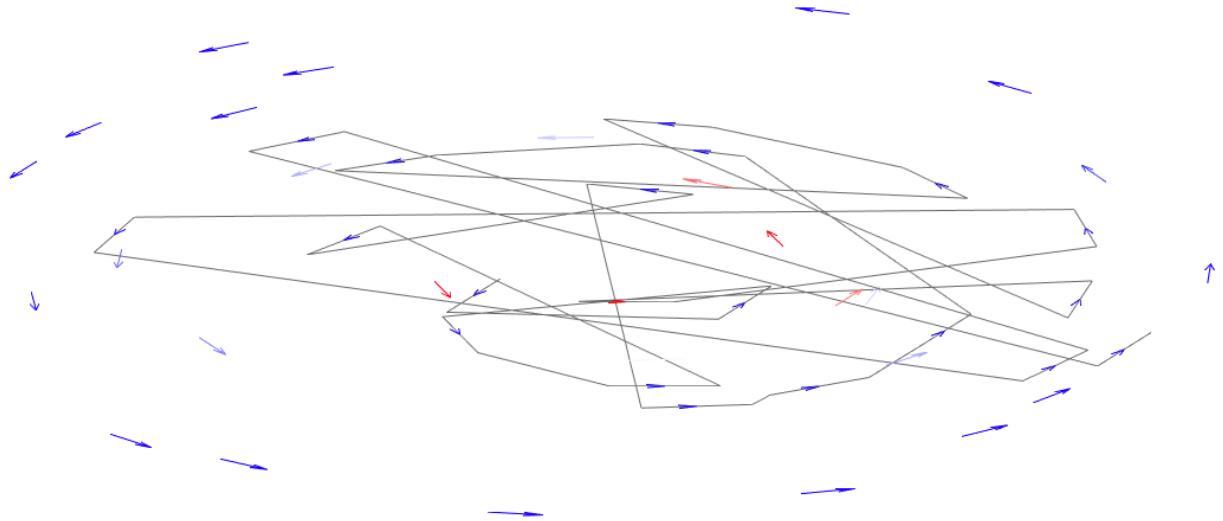


Figure 3.8: CP3 connects the first 20 vectors from a 20,000 vectors sample

5. If no vector's tail having the same height as current vector's head, go to the next layer and find the next closest vector until reach the top of the vector field.

6. Close the knot by connecting the head of last vector and the tail of first vector.

Detailed Explanation

NOTE: All three protocol start with converting the coordinate system to cylindrical coordinates. For the purpose of efficient grouping of vectors, *dictionary* (*unordered_map* in $C++$) is used as the data structure of storing vectors in the same layers (same z values).

Protocol 1

- In Step 4, if we start with a vector $v_i = (\theta_i, \rho_i, z_i)$, the next vector will be $v_{i+1} = (\theta_{i+1}, \rho_{i+1}, z_{i+1})$ such that

$$\theta_{i+1} = \min_{\theta > \theta_i} \theta$$

$$z_{i+1} = z_i$$

If there are multiple θ having the same value, the vector with smaller ρ will be selected. Based on the above conditions, a unique vector can be determined as the next vector

to be connected or no such vector exists.

- In Step 5, the next layer will have

$$z = \min_{z > z_i} z$$

And the first vector will have

$$\theta_0 = \min \theta$$

Same as before, if multiple vectors have the same θ, the one with smallest ρ will be selected.

Protocol 2

- In Step 3, the rationale behind is that there exist jumping vectors, which have different heights of heads and tails, in the vector field. Therefore, if fortunate enough, after connecting all vectors in the same layer, the jumping vectors will force connection to another layer.

- In Step 4, this protocol improves from Protocol 1 that when reach next layer z_{i+1}, current vector v_i does not connect the vector with smallest θ in layer z_{i+1}, but the one of closest θ with θ_i

Protocol 3

- In Step 4, this protocol improves from Protocol 2 that it avoids the situation connecting two vectors far apart even though they have close θ. In other words, it does not connect two vectors where spherical coordinates are far from Euclidean.

- To determine which vectors would be counted as "close", there are two different ways to choose the tolerance (Note: tolerance determines the range of radius of possible next vectors).

  - **Tolerance method 1 (Tm1):** The first way is to use a larger tolerance for vectors with radius greater than a threshold, and a smaller radius for the rest. The reason behind this is that we assume in the model there is a disordered core in the capsid, and an ordered structure at the capsid wall.
  - **Tolerance method 2 (Tm2):** The second is to use different tolerance for every vector based on their radius. The tolerance increases linearly with the radius of each vector.

- In Step 5, this protocol ensures the most number of layers are connected. It is a similar technique used in Step 5 in Protocol 1.

3.5 Compute Knot Type

In this section, we will briefly describe how to determine the knot type of the trajectories generated by using the Alexander polynomial.

In the programs, we start with projecting the trajectory to a plane from random angles. The next step is to store the information of intersection of segments in a matrix m and see how many crossings there are in the projection. If two segments cross with each other, we label the segment above with a +1 while the one below is labeled -1. Then we look for a +1 in m which is right handed and number it as +1 in the new matrix M. And we travel through m to obtain the dowker code of the knot and store the information in M.

With the dowker code available, we attempt to do the Reidemeister moves for the knot to simplify the structure. If after the simplification there are fewer than or equal to 2 crossings, we know that it is an unknot. If not, we compute the alexander polynomial as in Chapter 2 from the dowker code of the knot, and we replace the variable t with -1. When a knot is knotted, we know the value of the alexander polynomial is not 1.

Chapter 4

Results

4.1 Microvectors can be generated around the macrovector with small error

We have successfully generated sets of microvectors for a macrovector, with different number of microvectors shown in Figure 4.1 and different p values in Figure 4.2. In addition, different initialization values to be put in the Newton step or CG step yield different results. Tables containing different information are shown below. All tables list the size of microvectors, the objective value of the minimization problem, the standard deviation of $xyz-$coordinates of the microvectors, the p value and its related maximal angle between the microvectors and the macrovector, and the initialized range of x_0 for solving the minimization problem.

In Table 4.1, we change the number of microvectors generated from 25 to 100. The objective value which indicates the sum of Euclidean distance from microvectors to macrovectors square is decreasing from sizes 50 to 100. The standard deviation is also decreasing. These mean that as we increase the number of microvectors generated in the minimization, the microvectors are pointing more to the same direction. The angles between microvectors and the macrovector is set to be less than $\pi/3$. The initial $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are uniformly distributed random numbers with xyz-coordinates of the macrovector as the center, with range 0.01.

In Table 4.2, we change the p values and their corresponding θ of the microvectors. The entries are ordered in increasing p value and decreasing θ (microvectors are pointing in a smaller angle deviated from the macrovector). Note the objective values and standard deviations do not decrease with the angle. Instead, as we decrease the maximal angle allowed, the microvectors generated are more variant and deviate more from the macrovectors.

Table 4.1: Different sizes of microvectors and the properties of solutions to the minimization problem

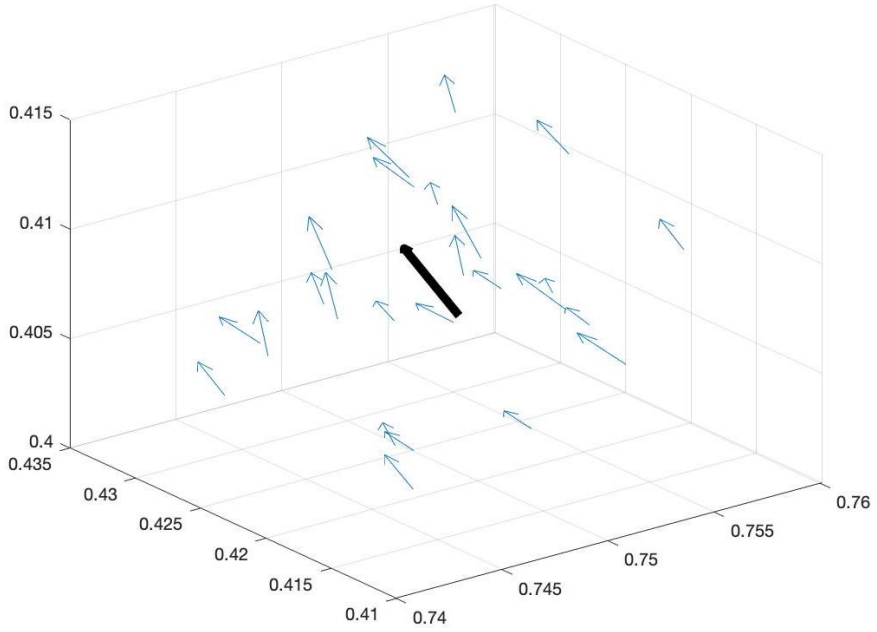
Size	Objective Value	Standard Deviation [x y z]			p	θ	Init
25	0.000020	[0.000585	0.000185	0.000649]	0.5	$\pi/3$	0.01
50	0.001191	[0.002833	0.000875	0.003918]	0.5	$\pi/3$	0.01
75	0.000853	[0.002361	0.000738	0.002311]	0.5	$\pi/3$	0.01
100	0.000007	[0.000176	0.000055	0.000174]	0.5	$\pi/3$	0.01

Table 4.2: Different maximal angles between the macrovector and the microvectors and the properties of solutions to the minimization problem

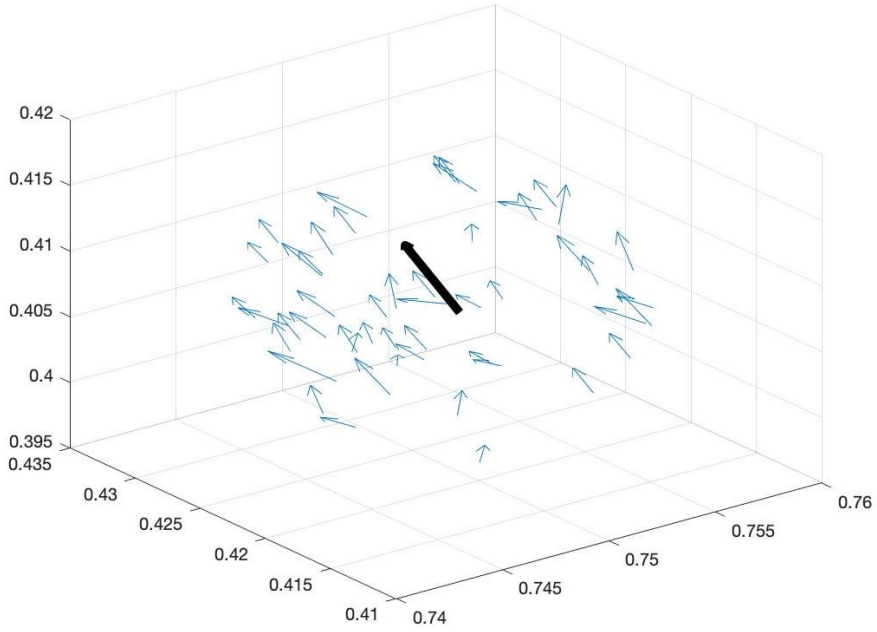
p	θ	Size	Objective Value	Standard Deviation [x y z]			Init
0.00	$\pi/2$	50	0.00000008	[0.00002384	0.00000751	0.00002810]	0.01
0.10	1.4706	50	0.00000883	[0.00025951	0.00008165	0.00031489]	0.01
0.30	1.2661	50	0.00021717	[0.00131604	0.00041966	0.00155609]	0.01
0.50	$\pi/3$	50	0.00064507	[0.00183404	0.00059065	0.00301664]	0.01
0.70	0.7954	50	0.00000018	[0.00003448	0.00001084	0.00004326]	0.01
0.90	0.4510	50	0.01962861	[0.01112294	0.01242454	0.01067291]	0.01

In Figure 4.1, the microvectors are generated with maximal angle $\pi/3$, and initialized range of x_0 for the minimization problem is 1.2. The objective values of (a) is 0.66275626; (b) is 4.57874648; (c) is 6.80597051; (d) is 6.69932780.

In Figure 4.2 and Figure 4.3, all graphs are showing 50 microvectors around the macrovector. The initialized range of x_0 for the minimization problem is 1.2. Still, the objective value indicates the differences in orientations of the microvectors. In Figure 4.2, the maximal angle allowed in (a) is $\theta = \pi/2$, objective value = 0.00000238; (b) is $\theta = \pi/3$, objective value = 3.47251175; (c) is $\theta = 0.4510$, objective value = 6.93049420. These figures again illustrate that the microvectors are more variant when we decrease the maximal angle θ allowed



(a)

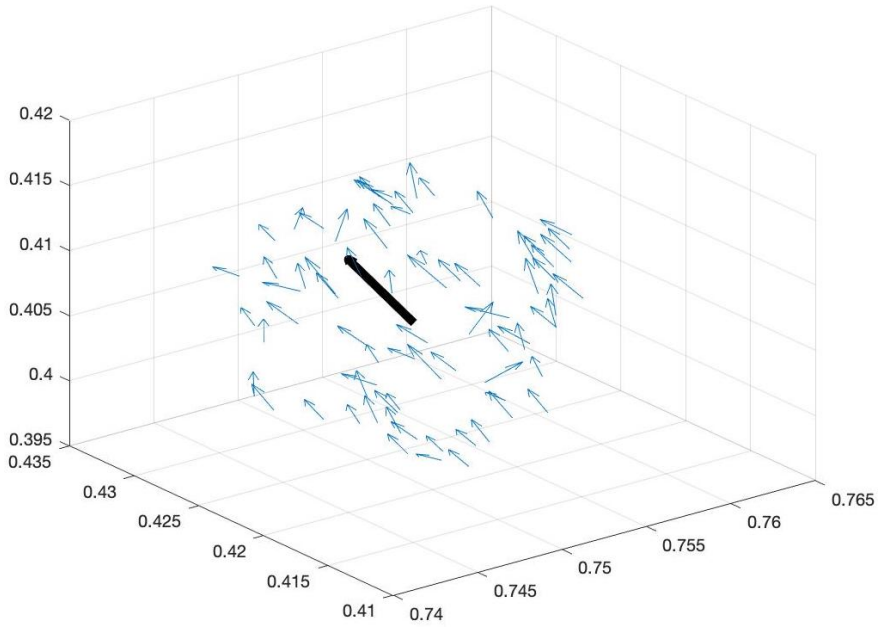


(b)

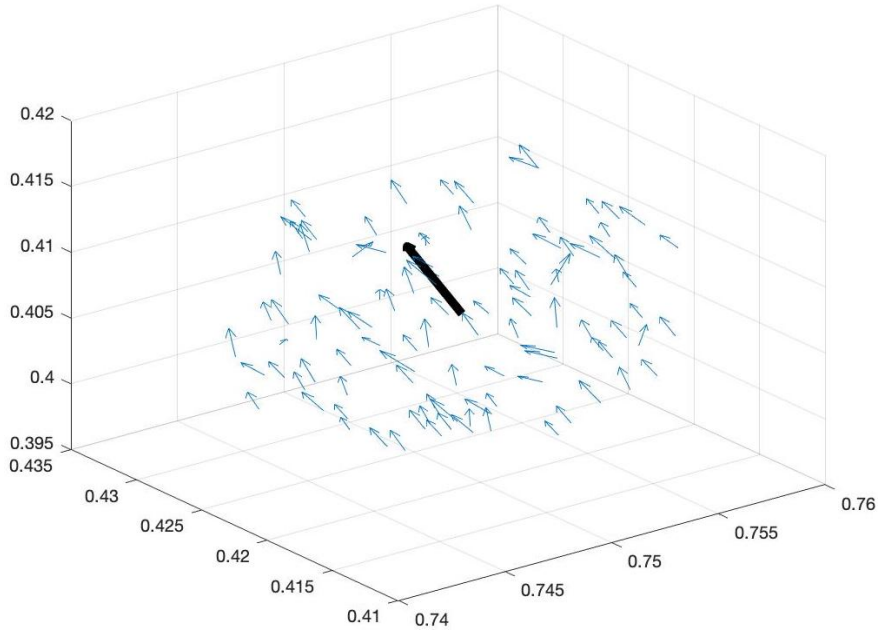
Figure 4.1

between the microvectors and the macrovector in the minimization problem.

In Table 4.3, we change the initialized range for $\mathbf{x}_0$ in the Newton step or conjugate gradient step. It is obvious that the objective value decreases as the range decreases. This is because if the starting point is close enough to the optimal solution, the objective value will



(c)



(d)

Figure 4.1: The graphs show the positions and orientations of different number of microvectors (blue, thinner) around the macrovector (black, thicker): (a) 25 microvectors; (b) 50 microvectors; (c) 75 microvectors; (d) 100 microvectors.

be closer to zero under the same number of iterations and tolerance limit. But this also raises another problem. If we start with a range of 0.00001 as in the table, the standard deviation of $\mathbf{x}, \mathbf{y}, \mathbf{z}$ is also too small to argue whether all microvectors are the same or distinct.

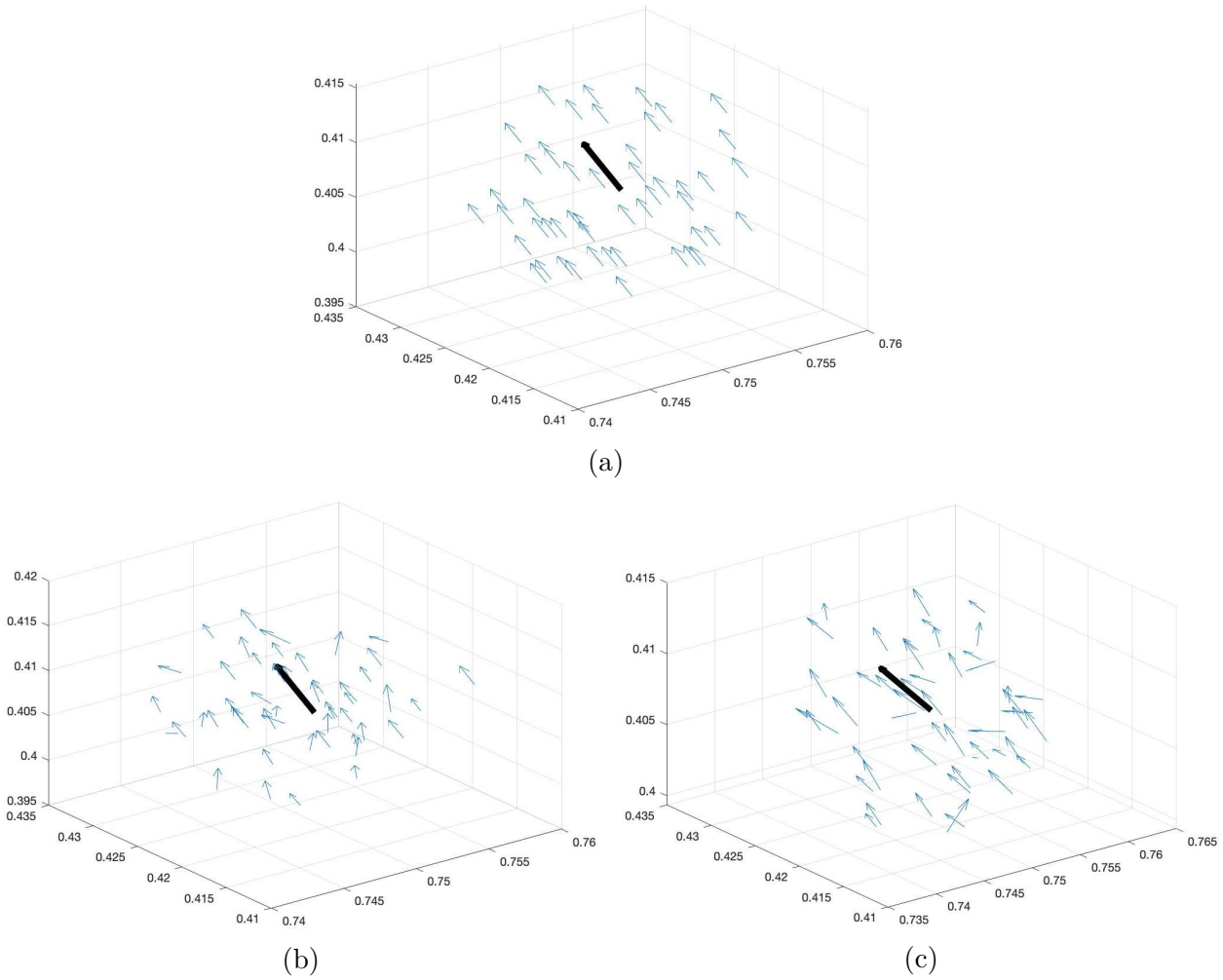
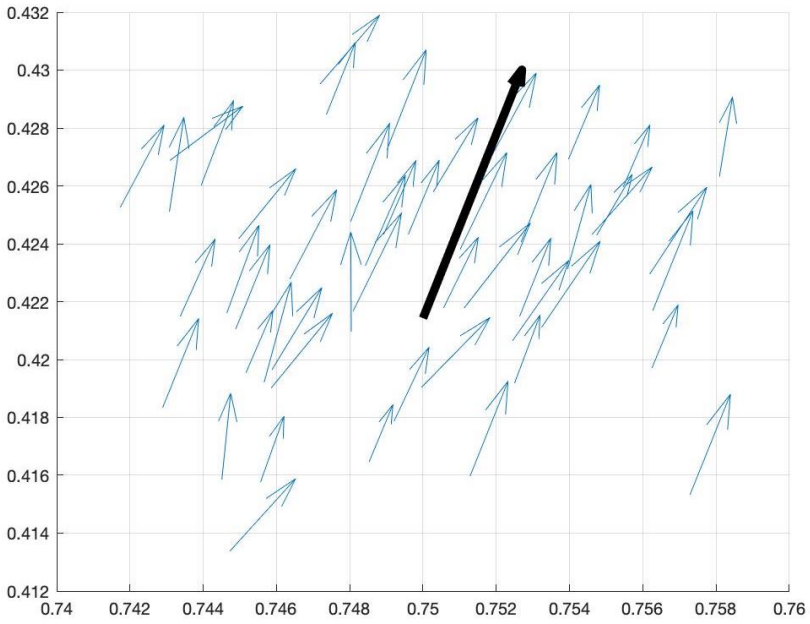


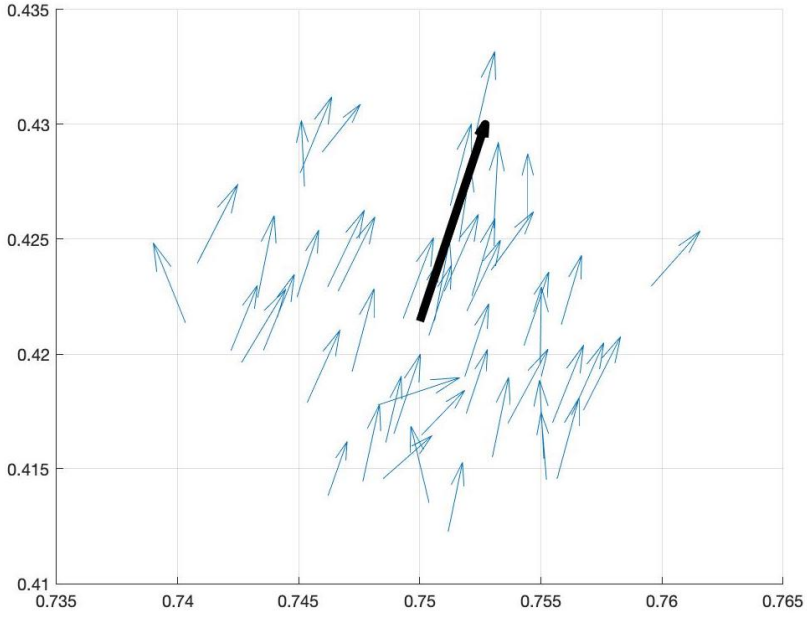
Figure 4.2: The graphs show the positions and orientations of three sets of microvectors with different maximal angles allowed in the minimization problem. (a) $\theta = \pi/2$; (b) $\theta = \pi/3$; (c) $\theta = 0.4510$.

Table 4.3: Different initialized ranges of x_0 in the minimization problem for generating microvectors.

Init	Size	Objective Value	Standard Deviation			p	θ
10.000000	50	57.5049852678	[0.51337117	0.67053608	0.38910776]	0.50	$\pi/3$
2.000000	50	15.7973021541	[0.36009198	0.15796036	0.39089375]	0.50	$\pi/3$
1.200000	50	2.2692157738	[0.08728974	0.11256847	0.14989586]	0.50	$\pi/3$
1.000000	50	1.6175858364	[0.10059184	0.11676156	0.08332003]	0.50	$\pi/3$
0.100000	50	0.0000110323	[0.00030505	0.00009589	0.00034347]	0.50	$\pi/3$
0.010000	50	0.0000035291	[0.00017616	0.00005529	0.00018710]	0.50	$\pi/3$
0.001000	50	0.0000001538	[0.00004326	0.00001361	0.00003182]	0.50	$\pi/3$
0.000100	50	0.0000000051	[0.00000618	0.00000193	0.00000348]	0.50	$\pi/3$
0.000010	50	0.0000000024	[0.00000006	0.00000009	0.00000003]	0.50	$\pi/3$



(a)



(b)

Figure 4.3: (a) is the vertical view of Figure 4.2(b); (b) is the vertical view of Figure 4.2(c).

4.2 Uniform sampling of the vector field is obtained

From our sampling method mentioned previously, it is possible to obtain a subsample of vectors distributed uniformly inside the vector field without losing generality. Then we can see some successful subsample in Figure 4.5. From Figure 4.5a to 4.5e, we can see that sample with less than 20,000 vectors is insufficient to represent the complete dsDNA filament inside the bacteriophage capsid. Based on these figures, we should always consider sampling more than 5000 vectors in order to get a more accurate model of dsDNA with liquid crystal.

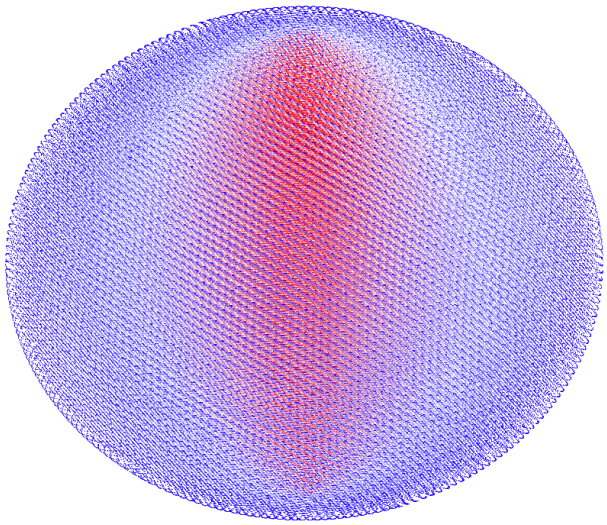
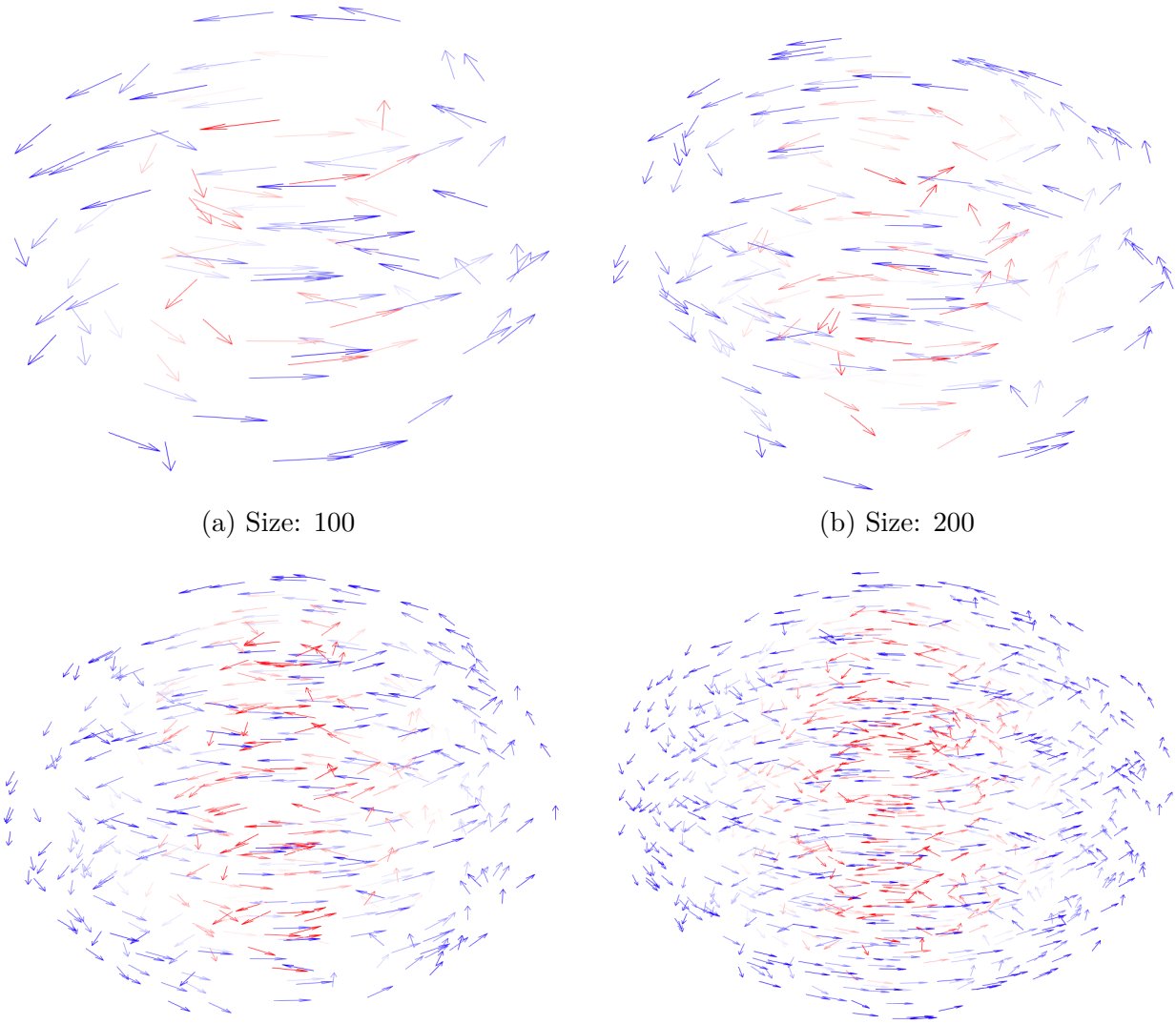


Figure 4.4: Visualizations of the original vector field

To compare these sampling results to our original vector field, visualizations of the original vector field is presented in Figure 4.4.



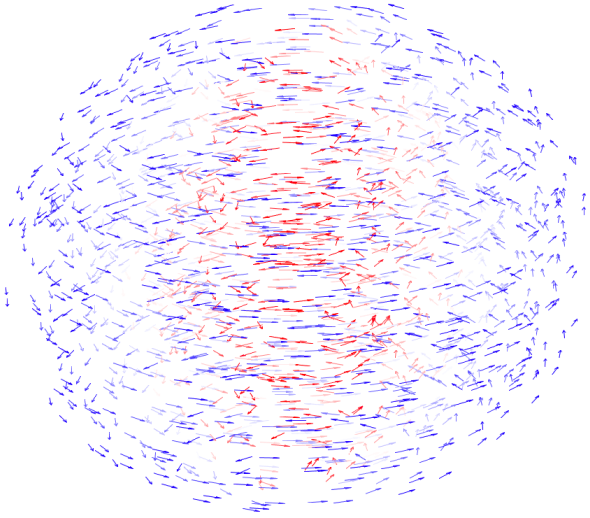
(a) Size: 100

(b) Size: 200

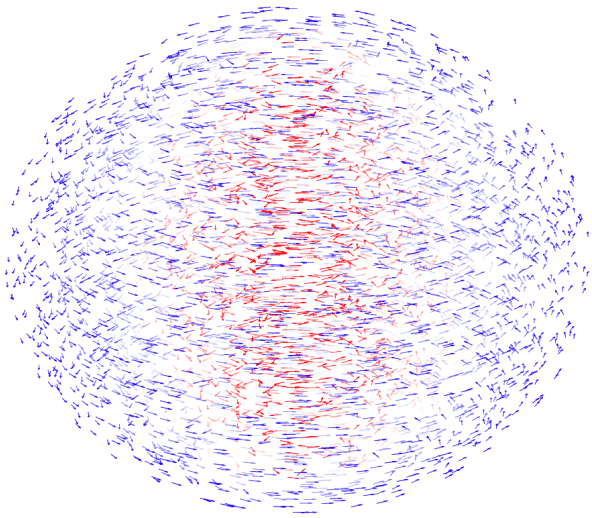
(c) Size: 500

(d) Size: 1,000

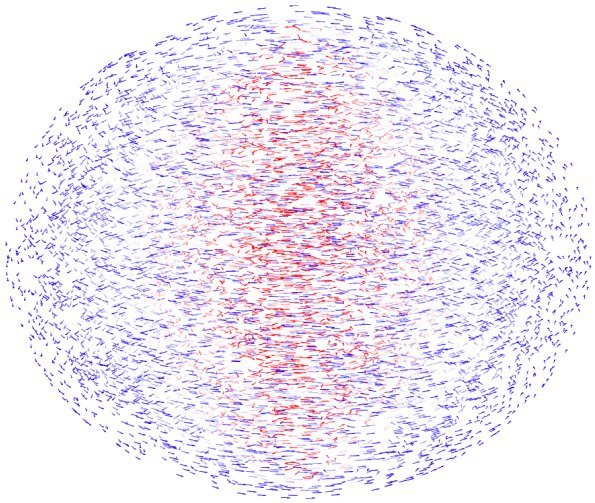
Figure 4.5: Different sample sizes visualization



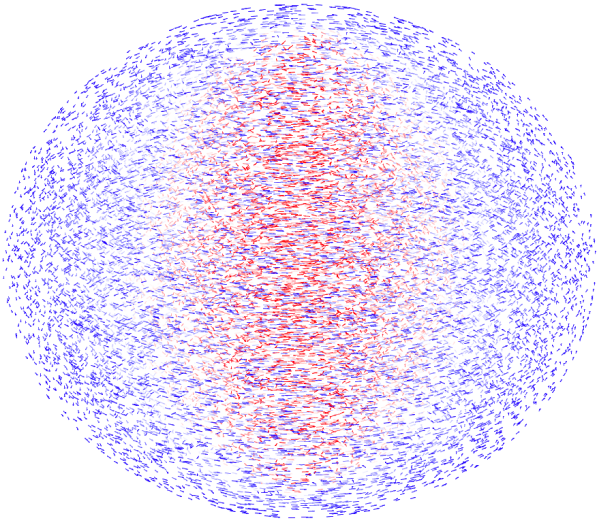
(e) Size: 2,000



(f) Size: 5,000

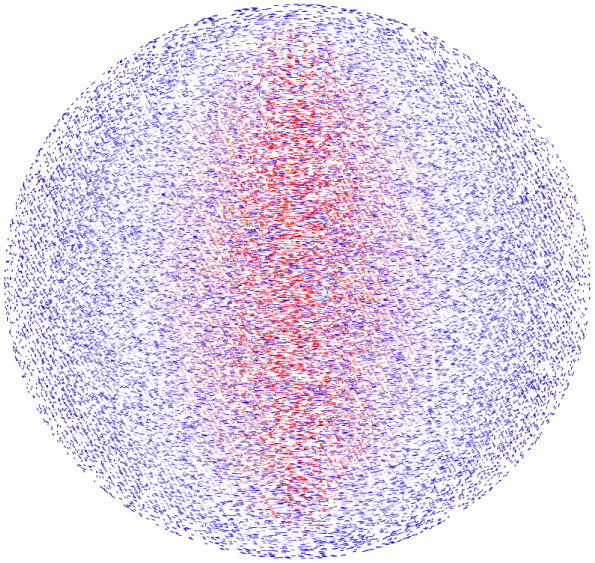


(g) Size: 10,000

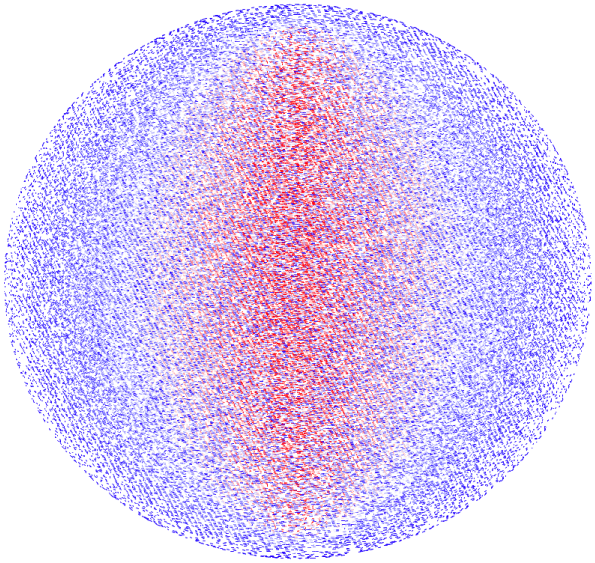


(h) Size: 20,000

Figure 4.5: Different sample sizes visualization (cont.)



(i) Size: 50,000



(j) Size: 100,000

Figure 4.5: Different sample sizes visualization (cont.)

4.3 Different Trajectories Are Obtained From Different Connecting Protocols And Parameters

For the three connecting protocols proposed in the previous chapter, it is not possible to determine which one is the completely correct because the structure inside the bacteriophage capsid is still unknown. In this section, the advantages and limitations of each connecting protocol (CP) will be discussed.

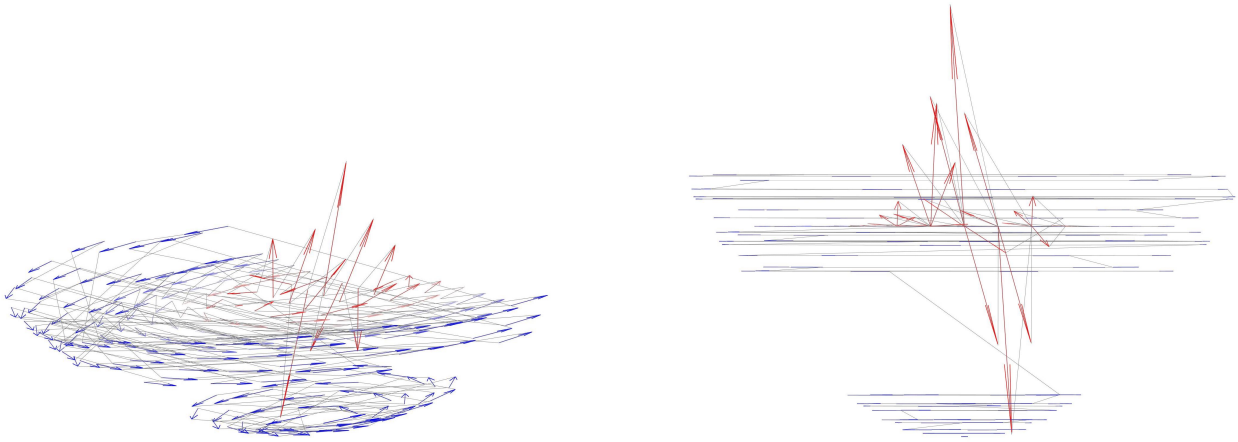
CP1:

Advantages:

- Every vector in the vector field is connected, which means all layers are connected even when there is only one vector per layer.
- It is easy to implement because this protocol does not need to care about how to jump between layers.

Limitations:

- The jumping vectors do not change layers because the first task would be to connect all vectors with same height of tail. In Figure 4.6a, the red vectors represent jumping vectors. It is shown that these vectors need to go back to the layers of its tail even when it is obviously pointing to a different layer (height). In other words, the jumping vectors are not working in this protocol.
- Because when changing layers, the vector with smallest θ is always connected. It could be the case that two vectors far apart will be connected even when there exists a closer vector in terms of θ. Figure 4.6b shows how layers are changed when all vectors in the same layer are connected. It also illustrates the problem discussed before that jumping vectors are inactive for changing layers.



(a) Connecting the first 200 vectors (b) Connecting the first 80 vectors

Figure 4.6: Connecting first 80 and 200 vectors with CP1

CP2:

Advantages:

- This protocol is a huge improvement from **CP1** because it recognizes the heights of heads and tails and is able to put them into different groups.
- Changing layers would only occur when connecting to the jumping vectors. Therefore, it is possible that the first and last vectors are near, and it is the desired situation for closing the knot.
- The closer vectors in term of θ in new layers in connected.

Limitations:

- The connection is highly dependent on the sampling results. If the sample consists of a larger proportion of jumping vectors, more layers are visited. But if only a few jumping vectors are available, the connection could end up early with only a small number of vectors connected.
- The connection also relies on how well the first vector is randomly selected. If the first vector is a jumping vector, the chance that vectors in the tail layer of jumping vector

being discovered is low. And due to the property that most jumping vectors are in the center of the structure, the connection may not discover the vectors in the periphery, and therefore the structure is incomplete.

- In Figure 4.7, it is shown that a large proportion of vectors in the periphery are not connected (by gray lines). By this protocol, many vectors near the outer region are not included in the trajectory.

- It is also possible that two close vectors in terms of θ are not close in space if their Euclidean distances r to the $z-$axis varies hugely. This possibility leads to the situation in Figure 4.8 if look at the center.

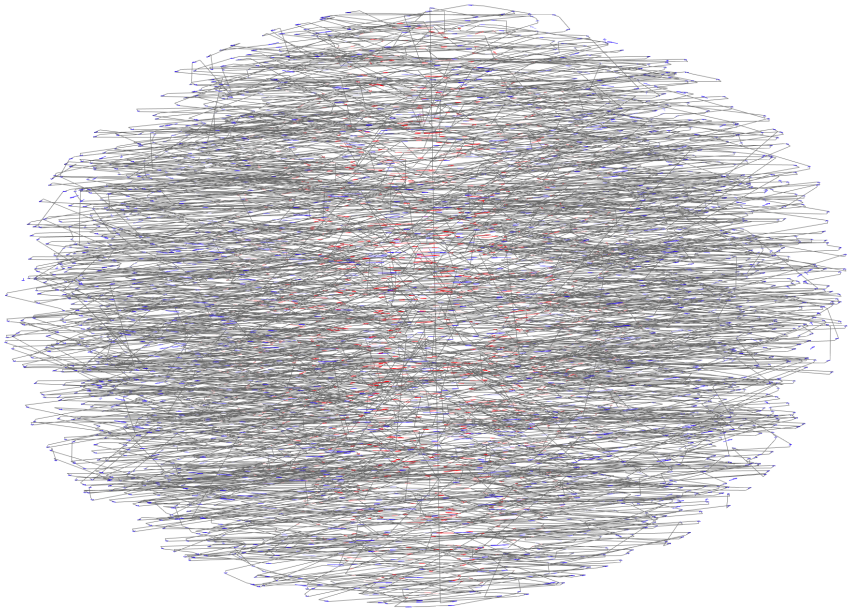


Figure 4.7: Connecting 5,000 vectors with CP2

CP3:

Advantages:

- This protocol incorporates the advantages from **CP1** and **CP2** that it can connect as many layers in the structure as possible, and changes layers more reasonably because

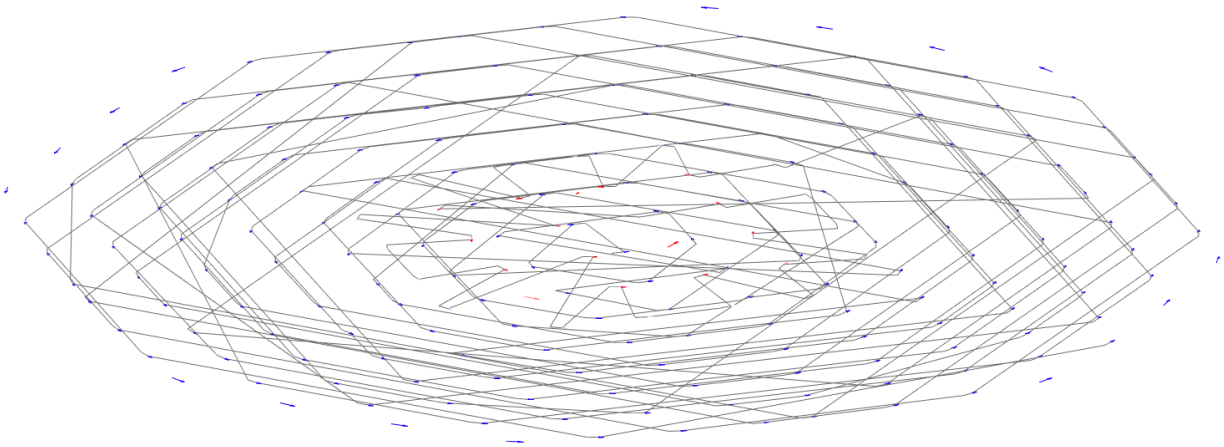


Figure 4.8: Connecting first 200 vectors in CP2

it can automatically select vectors near not only in term of θ but also r, which solves the last limitation mentioned in **CP2**.

- We can control manually the tolerance for a more reasonable local connection, as shown in Figure 4.9.

Limitations:

- Because the vectors always connect from the bottom to the top, it is impossible that the first and last vectors are close in space.
- Restricting the distance from current vector to the next will also reduce the number of points available to connect.

In Table 4.4 and Table 4.5, we present how different connecting protocols (CP) act on different sizes of subsamples. Because CP1 has bad performance in connecting samples from the vector field, we do not include the result of this protocol in this table. Here, the numbers and percentages are the averages of 10 experiments of each protocols. Tolerance method 1 refers to using one tolerance value (next to CP3 in the first column) for the ordered region and another value (1/5 of the ordered tolerance) for the disordered inner core (radius = 0.1). In Tolerance method 2, we choose different tolerance for each vector according to its distance to the z-axis. The tolerance increases linearly with the radius.

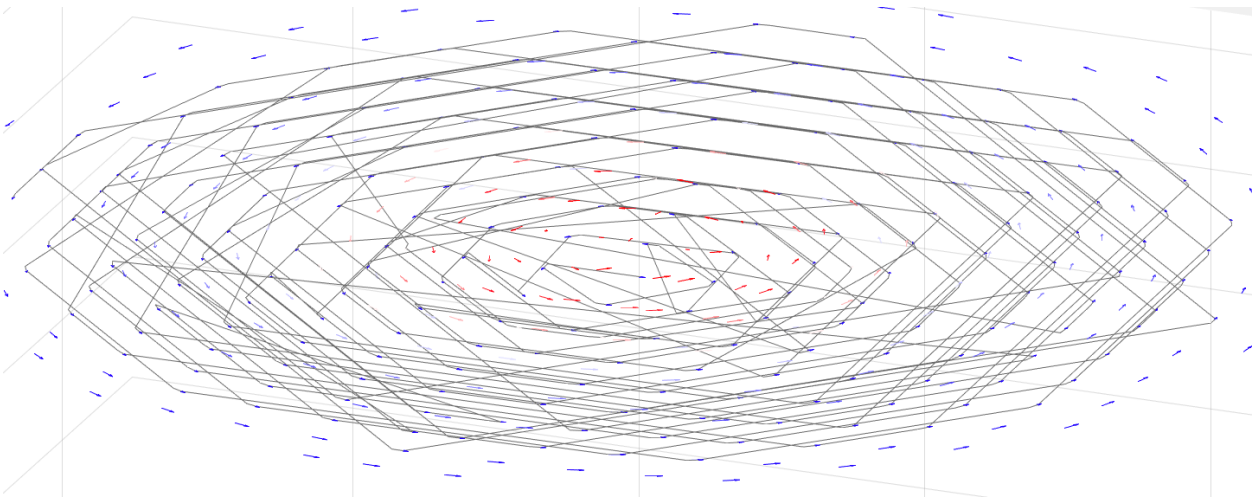


Figure 4.9: Visualization of the trajectory generated after connecting the first 200 vectors using CP3, Tm 1, tol=0.005. However, it will be shown in Table 4.4 that such value of tol is not reasonable. But this avoids the last limitation of CP2.

Comparing Table 4.4 and Table 4.5, we can see that more vectors are used in most cases if we do not allow jumping between layers. This is due to the fact that if jumping vectors are met early in a layer, jumping to another layer would omit the vectors left out in the previous layer.

Based on Table 4.4 and Table 4.5, we can conclude the following:

- CP2 is more advantageous than CP3 for most sizes of samples if the primary goal is to use as many vectors as possible.
- Tolerance method 2 uses more vectors than Tolerance method 1 with other parameters being the same.
- The number of vectors used in the trajectory decreases as we decrease the tolerance in CP3. This is because smaller regions are allowed when the algorithm looks for next possible vector from current position.
- When the sample size goes over 10,000, the percentages of vectors use are less than 10% with any protocol and tolerance, except for some cases in samples of 100,000 vectors.

However, Tolerance method 2 is not perfect. Because of the extremely small tolerances

Table 4.4: Vector usage information with different sample sizes and protocols (Jumping Layer Allowed)

Each column shows the average number and percentage of vectors used in different sample sizes. The first row states the sizes of sample from the vector field. In the following rows, CP2 means Connecting Protocol 2; CP3 means Connecting Protocol 3, and the value next to it is the tolerance for vector with radius 0.5.

Sample size	100	200	500	1000	2000	5000	10000	20000	50000	100000	191489 (ALL)
Average Numbers of Vector Used in the Connecting Protocols											
CP2	83.00	137.70	227.30	309.30	450.40	853.40	1014.10	1864.60	3670.70	15612.80	16580.00
Tolerance method 1											
CP3 0.5	80.20	134.40	216.40	300.20	432.20	792.00	1272.40	1960.10	3141.90	8953.30	15629.00
CP3 0.2	73.90	111.20	176.10	263.60	388.60	802.90	1122.90	1716.80	3385.20	16386.80	8983.00
CP3 0.1	69.70	104.00	151.30	211.40	316.80	573.50	1121.10	1741.60	2788.20	6911.00	8162.00
CP3 0.05	68.90	100.10	145.30	188.50	273.90	511.80	788.20	1372.40	1962.50	6728.00	6481.00
CP3 0.01	70.20	97.20	139.00	183.90	234.50	476.90	706.30	1031.30	2031.80	2852.50	3456.00
Tolerance method 2											
CP3 0.5	82.70	136.50	224.50	308.80	457.70	861.00	1289.30	1838.50	3933.50	12027.90	16439.00
CP3 0.2	76.00	128.80	210.40	289.20	431.80	768.70	1303.70	1166.80	3435.70	19466.40	12103.00
CP3 0.1	75.90	115.60	187.10	261.10	367.50	807.30	1162.00	1784.90	3362.60	13711.20	9484.00
CP3 0.05	71.70	107.20	164.00	227.10	300.40	648.20	1097.30	1848.70	2555.50	14408.00	8668.00
CP3 0.01	69.30	99.30	143.10	189.20	264.40	439.60	649.40	1340.70	1877.00	5604.00	5299.00
Percentages of Vector Used (%)											
CP2	83.00	68.85	45.46	30.93	22.52	17.07	10.14	9.32	7.34	15.61	8.66
Tolerance method 1											
CP3 0.5	80.20	67.20	43.28	30.02	21.61	15.84	12.72	9.80	6.28	8.95	8.16
CP3 0.2	73.90	55.60	35.22	26.36	19.43	16.06	11.23	8.58	6.77	16.39	4.69
CP3 0.1	69.70	52.00	30.26	21.14	15.84	11.47	11.21	8.71	5.58	6.91	4.26
CP3 0.05	68.90	50.05	29.06	18.85	13.70	10.24	7.88	6.86	3.92	6.73	3.38
CP3 0.01	70.20	48.60	27.80	18.39	11.72	9.54	7.06	5.16	4.06	2.85	1.80
Tolerance method 2											
CP3 0.5	82.70	68.25	44.90	30.88	22.89	17.22	12.89	9.19	7.87	12.03	8.58
CP3 0.2	76.00	64.40	42.08	28.92	21.59	15.37	13.04	5.83	6.87	19.47	6.32
CP3 0.1	75.90	57.80	37.42	26.11	18.38	16.15	11.62	8.92	6.73	13.71	4.95
CP3 0.05	71.70	53.60	32.80	22.71	15.02	12.96	10.97	9.24	5.11	14.41	4.53
CP3 0.01	69.30	49.65	28.62	18.92	13.22	8.79	6.49	6.70	3.75	5.60	2.77

for vectors near the center axis and the fact that jumping vectors are cumulating around the center axis, the connection could ignore some of them. Therefore, the curve is forced to go to the next layer after connecting all vectors in the same layer, possibly without using (or skipping) the jumping vectors in the same layer.

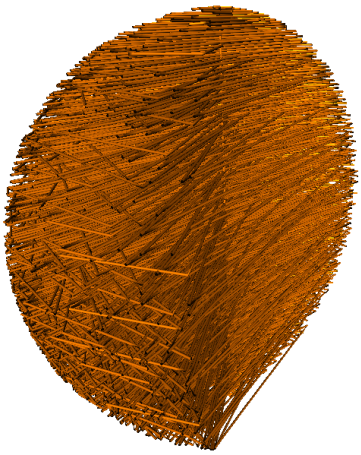
Based on the conclusion in the previous section that sample size should be greater than 5,000, we visualize some trajectories with 20,000, 50,000, and all vectors, in different CP's and Tm's.

In the next few pages, the graphs illustrates how different CP's and different Tolerance

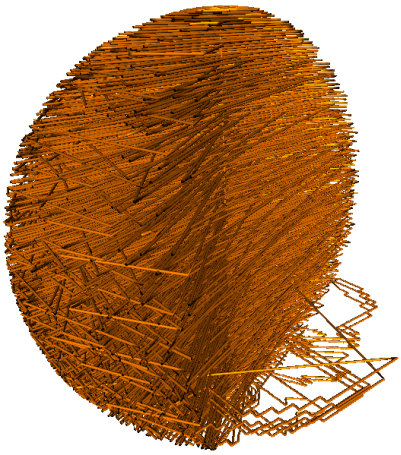
Table 4.5: Vector usage information with different sample sizes and protocols (Jumping Layer NOT Allowed)

Sample size	100.00	200.00	500.00	1000.00	2000.00	5000.00	10000.00	20000.00	50000.00	100000.00	191489 (ALL)
Average Numbers of Vector Used in the Connecting Protocols											
CP2	99.10	196.00	478.50	930.20	1759.20	4012.40	7616.90	15822.20	33306.80	77695.30	133339.00
Tolerance method 1											
CP3 0.5	94.20	180.20	438.10	806.80	1440.40	3302.10	6719.50	12346.10	28527.70	61310.60	128232.50
CP3 0.2	78.30	126.50	249.60	426.10	944.40	2520.00	4536.30	7111.50	18423.40	38560.20	110780.50
CP3 0.1	70.90	109.50	158.00	244.00	409.70	1038.90	2894.30	7446.60	14330.10	22376.20	61774.50
CP3 0.05	71.70	99.20	146.30	200.80	287.00	562.00	1013.20	1982.70	7973.60	14498.80	13984.50
CP3 0.01	68.80	98.40	140.30	182.50	260.90	472.30	699.00	1298.30	1845.80	2612.90	3455.50
Tolerance method 2											
CP3 0.5	98.50	195.10	477.70	924.80	1717.10	4059.20	7819.50	14474.60	31414.40	73375.70	133482.50
CP3 0.2	90.40	172.50	400.50	812.60	1587.60	4074.70	7482.90	14934.30	33611.90	59871.00	130999.50
CP3 0.1	80.60	141.60	267.30	475.90	898.10	2791.00	6024.40	12550.70	31087.70	52144.50	123409.50
CP3 0.05	72.80	113.90	193.40	278.50	476.10	1077.40	2400.70	6160.20	23129.60	37155.80	101059.50
CP3 0.01	67.00	99.10	144.70	192.30	276.30	504.10	824.70	1477.90	2974.70	6792.80	16966.50
Percentages of Vector Used (%)											
CP2	99.10	98.00	95.70	93.02	87.96	80.25	76.17	79.11	66.61	77.70	69.63
Tolerance method 1											
CP3 0.5	94.20	90.10	87.62	80.68	72.02	66.04	67.19	61.73	57.06	61.31	66.97
CP3 0.2	78.30	63.25	49.92	42.61	47.22	50.40	45.36	35.56	36.85	38.56	57.85
CP3 0.1	70.90	54.75	31.60	24.40	20.48	20.78	28.94	37.23	28.66	22.38	32.26
CP3 0.05	71.70	49.60	29.26	20.08	14.35	11.24	10.13	9.91	15.95	14.50	7.30
CP3 0.01	68.80	49.20	28.06	18.25	13.05	9.45	6.99	6.49	3.69	2.61	1.80
Tolerance method 2											
CP3 0.5	98.50	97.55	95.54	92.48	85.85	81.18	78.20	72.37	62.83	73.38	69.71
CP3 0.2	90.40	86.25	80.10	81.26	79.38	81.49	74.83	74.67	67.22	59.87	68.41
CP3 0.1	80.60	70.80	53.46	47.59	44.91	55.82	60.24	62.75	62.18	52.14	64.45
CP3 0.05	72.80	56.95	38.68	27.85	23.80	21.55	24.01	30.80	46.26	37.16	52.78
CP3 0.01	67.00	49.55	28.94	19.23	13.82	10.08	8.25	7.39	5.95	6.79	8.86

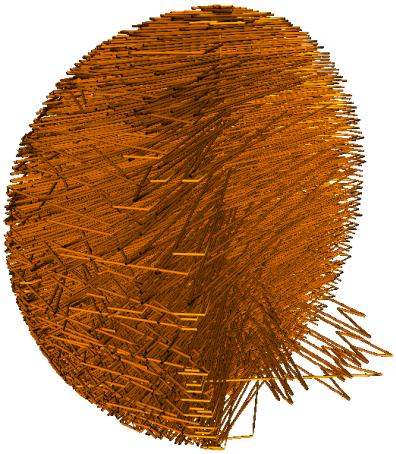
methods' in CP3 affect the number of vectors used and therefore the smoothness of trajectories. All trajectories are connected using macrovectors from the original vector field instead of microvectors. From Figure 4.10a to Figure 4.10f, we see that the structure of trajectories does not form a sphere. It means the DNA does not fill the entire capsid, which disagrees with our proposed model. Therefore, the figures imply that tolerance less than or equal to 0.2 should not be used in all sizes of samples because it is not coherent with the model. Comparing the left column (Tm1) and right column (Tm2), more vectors are used in the trajectories with Tolerance method 2. This also aligns the data and conclusion derived from Table 4.4.



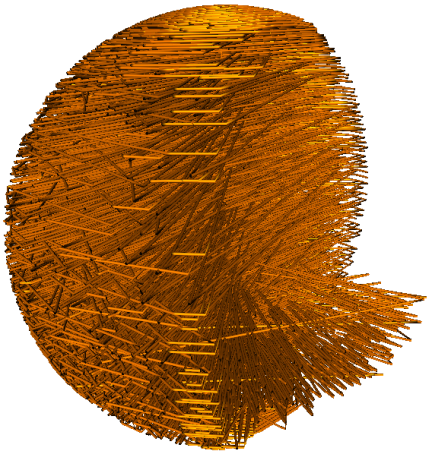
(a) CP3: All vectors, Tm1, Tol=0.01



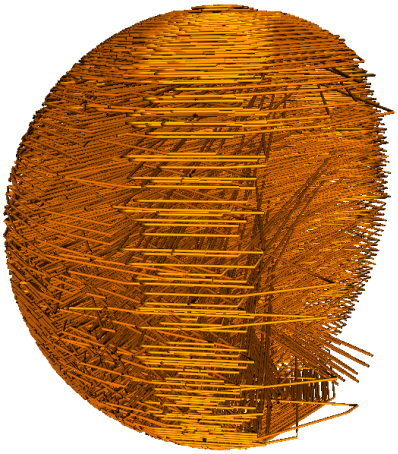
(b) CP3: All vectors, Tm2, Tol=0.01



(c) CP3: All vectors, Tm1, Tol=0.05

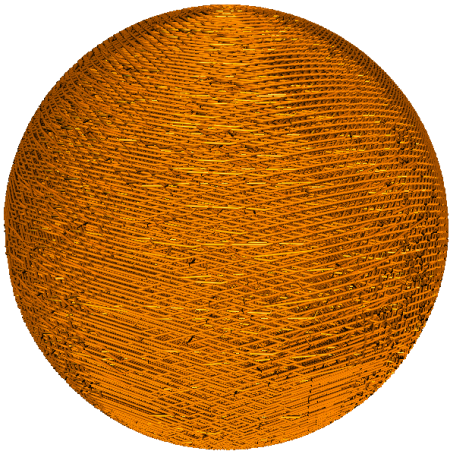


(d) CP3: All vectors, Tm2, Tol=0.05

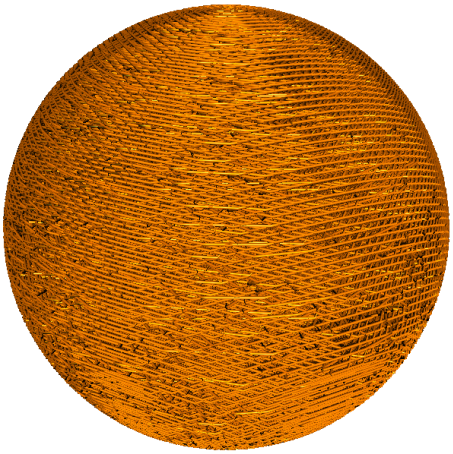


(e) CP3: All vectors, Tm1, Tol=0.20

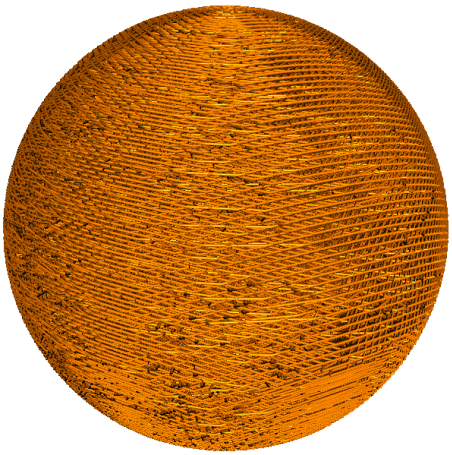
(f) CP3: All vectors, Tm2, Tol=0.20



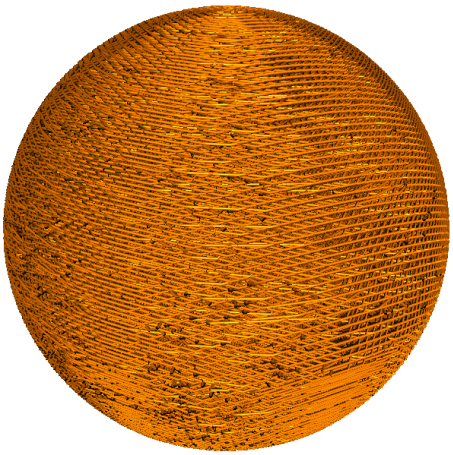
(g) CP3: All vectors, Tm1, Tol=0.50



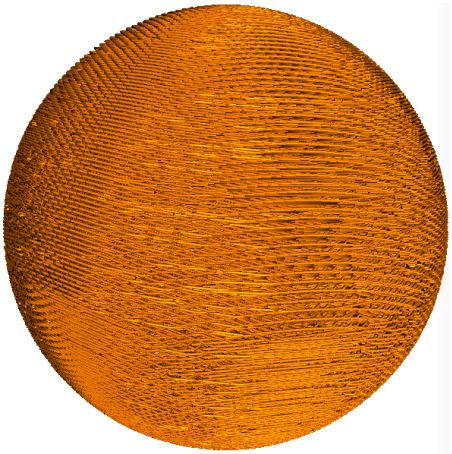
(h) CP3: All vectors, Tm2, Tol=0.50



(i) CP3: All vectors, Tm1, Tol=1.00



(j) CP3: All vectors, Tm2, Tol=1.00



(k) CP2: All vectors

(l) CP2:50,000 vectors

(m) CP3: 50,000 vectors, Tm1, Tol=0.50

(n) CP3: 50,000 vectors, Tm2, Tol=0.50

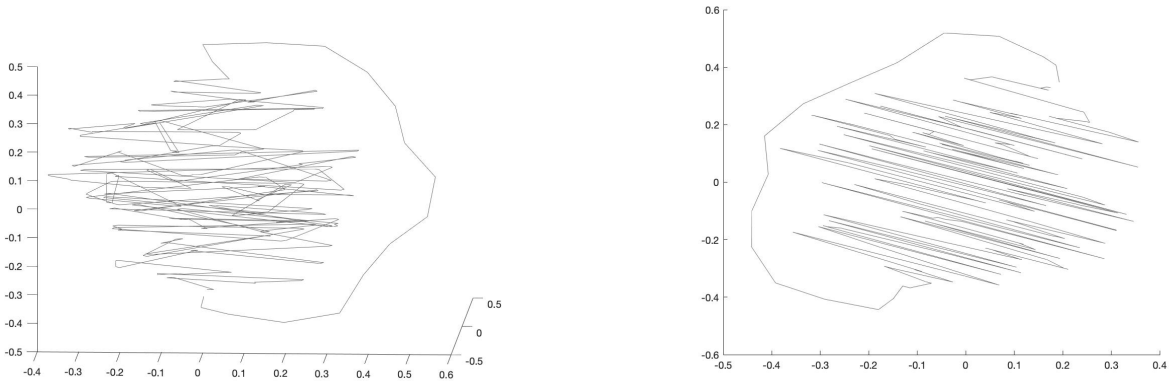
(o) CP3: 20,000 vectors, Tm1, Tol=0.50

(p) CP3: 20,000 vectors, Tm2, Tol=0.50

Figure 4.10: These graphs show the trajectories of DNA when using different protocols and parameters.

4.4 The Trajectories Are Unknotted

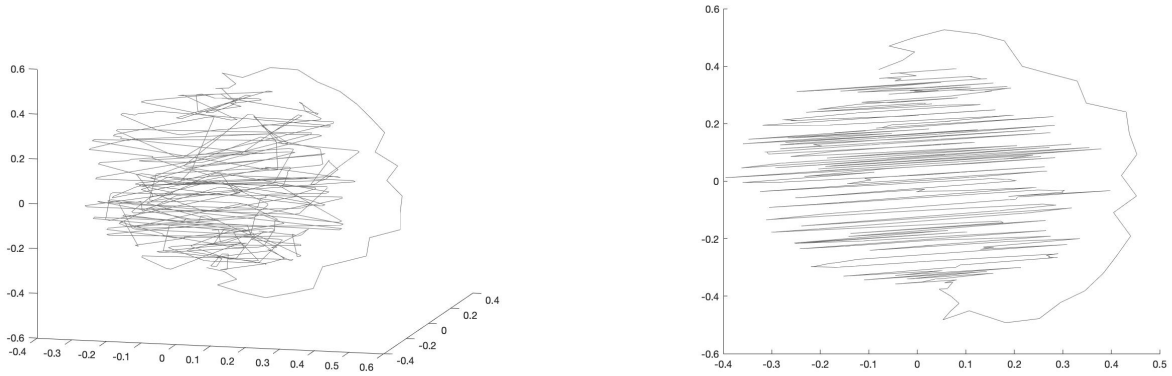
In this section, we will present the results of computing the knot type of the trajectories. More importantly, we need to see if they are knotted or not. Due to the limit of computation power of available devices, we tested the trajectories generated by different sample sizes from 50 to 500. They are all unknotted. Some projections are showed for specific sample sizes.



(a) Regular view of the trajectory

(b) Successful projection for computing the knot type

Figure 4.11: Different views of trajectory of connecting 150 vectors



(a) Regular view of the trajectory

(b) Successful projection for computing the knot type

Figure 4.12: Different views of trajectory of connecting 500 vectors

From Table 4.6, we see that trajectories generated from less than 500 macrovectors are unknotted, which disagrees with our experimental data. From experiments, we see more than

Table 4.6: Average number of crossings in projections for different trajectories

Number of vectors in the sample	Number of points	Average number of crossings	Knotted or not
50	107	149	**Unknotted**
100	176	469	
150	262	644	
250	304	964	
350	407	1476	
400	430	1665	
500	486	2007	
1000	619	2889	**Unknown**
5000	1722	16641	
20000	4522	117658	
All	35177	1560863	

90% are knots, but here we do not see any. The reasons behind it are (1) the trajectories are set to go up from the bottom to the top in all protocols; (2) the number of vectors in the reconstruction is small; (3) the jumping vectors in the core are ignored. Therefore, the trajectories always go up and changing between layers does not occur frequently.

Chapter 5

Conclusions and Future Work

The motivation of this thesis is to understand the knotting structures of trajectories generated by connecting the solutions to the PDEs.

Because the actual folding of DNA (trajectory) inside the viral capsid is unknown, even when we obtain the energy fields, we are unable to determine how a complete trajectory can be generated. We developed three different connecting protocols, which allow us to explore different scenarios. In addition, we successfully generate microvectors around the macrovectors with small error. This also enables us to obtain different trajectories by connecting microvectors instead. Due to the limit of computational power, a sampling method is also designed in order to do the computation of knotting in smaller sizes.

5.1 Limitation of the work

The main limitation of this work is that the trajectories we have generated are all connecting from the bottom end to the top end such that the starting point and ending point are far away. Because when DNA enters a capsid, it moves freely until all of it is inside the capsid, it is possible that the head and tail of the DNA connect with each other. Therefore, it is reasonable to formulate a connection protocol which the starting and ending points are close.

In addition, from Table 4.4, we can see that only a small portion of vectors are used in

the trajectories for most sample sizes and protocols when we decide to jump between layers. This limitation can be caused by not enforcing the trajectory to connect from the peripheral at each layer. And because most jumping vectors are located near the z-axis, or center of the vector field, when the process is jumping, it is more likely to stay in the center and keep jumping.

Another limitation is that there are still sharp changes of direction in the trajectories generated with small sample sizes, which means the curves are not smooth. This could be addressed by interpolating smooth curves between vectors.

Moreover, if we want the microvectors generated to be more variant (Figure 4.1), instead of pointing to nearly the same direction (Figure 4.2(a)), our current method would also give larger errors that the difference between the mean of microvectors and the macrovector is larger. In other words, more variant microvectors are less accurate.

Finally, we have only analyze the knotting properties of trajectories with sample size up to 500. The results are not accurate enough to make a conclusion on the complete trajectory of DNA because of the small sample size.

5.2 Future work

There are many directions for future work. One direction is to develop another connecting protocol such that starting at any random point in the vector field and ending up the connection at a close point to the starting position. Also, we can consider improving the sampling method according to the connecting protocols such that more vectors can be used in generating the trajectories. With such improvement, we will be able to derive a more complete and reliable trajectory using a smaller sample.

More trajectories can also be derived from using the microvectors in the connection. But we need to balance between the variances and the accuracies of microvectors generated. Therefore, it is possible to obtain them in different ways instead of formulating an

optimization problem.

Another way to analyze the knotting property can be done by speeding up the computation of knot type program. This allows us to understand the structures of more sophisticated trajectories of larger sample or even the one obtained from original vector field without sampling.

Chapter 6

Acknowledgements

I was lucky to have Javier Arsuaga as my advisor at UC Davis. It was him who gave the opportunity of doing research work as an undergraduate student to me. I took Javier's MAT 128B and after that we decided the work to do in this thesis. I also thank him for taking patience discussing problems with me every week and looking for resources on my level whenever I see new concepts and algorithms.

Amongst students current and former, thanks to Zihao Zhu for introducing this project to me and sharing ideas on writing the first few programs. Thanks also to Tamara M Christiani, Anthony Jajeh, and Wangbo Tang for giving advice to my thesis and presentation.

Thanks to my parent Xunren and Liping for instilling the respect and love for education, which made me so natural to continue in academia instead of going into industry right after my graduation from UC Davis.

This research was partially supported by a grant from the NSF: DMS- 1817156.

Appendices

Appendix A

Data Files

Triangulation of the spherical volume:TET Tetrahedra are numbered from 1 to 1103760 and each line in the file contains the vertex number corresponding to one tetrahedron.

Coordinates of the vertices:VTX Each row contains the coordinates for the vertices of a tetrahedron. The row of the coordinates corresponds to each of the columns in file TET.

Vectors:N Contains the n vector field and is mapped to the coordinates in VTX by row number. In other words the first vector in N is placed starting at the first point in VTX.

Disorder parameter:S It is mapped by row number as before and it contains the value of the disorder parameter.

95729	95723	95722	69
95730	95724	95723	70
95730	95731	95725	71
95736	95727	95726	81
95737	95728	95727	82
95738	95729	95728	83
95739	95730	95729	84
95739	95740	95731	85
95740	95741	95732	86
95741	95742	95733	87
95746	95735	95734	95

Figure A.1: Screenshot of data TET

0.459982364323622	0.474268078609337	0.102839507180765
0.459391592662529	0.487963021233958	0.102248735519672
0.459231363905370	0.497911493237487	0.102088506762513
0.459391592662529	0.512036978766042	0.102248735519672
0.459982364323622	0.525731921390663	0.102839507180765
0.474268078609337	0.459982364323622	0.102839507180765
0.473224873325315	0.473224873325315	0.101796301896744
0.472585176197812	0.486870890483527	0.101156604769241
0.472384148326178	0.499044423102394	0.100955576897606
0.472585176197812	0.513129109516473	0.101156604769241
0.473224873325315	0.526775126674685	0.101796301896744

Figure A.2: Screenshot of data VTX

-0.972740323519100	0.231896339461334	-5.922367687584176e-04
-0.977150799687150	0.212546731124898	-4.491757833318359e-04
-0.980574630902340	0.196146062183251	-3.398834277127330e-04
-0.983283747073461	0.182079672323487	-2.562571845215817e-04
-0.985463433649963	0.169887562948299	-1.920789244088363e-04
-0.987242988627240	0.159220793664263	-1.423772395289760e-04
-0.988714572143576	0.149811495480926	-1.032119867185872e-04
-0.989945299784240	0.141450692058119	-7.177304225026075e-05
-0.990985183824256	0.133971501759232	-4.644548002143311e-05
-0.991872411012297	0.127236471046276	-2.661472007680115e-05
-0.992636532733879	0.121130977637800	-1.177172729220275e-05

Figure A.3: Screenshot of data N

0.761296168756756
0.799990000000000
0.799990000000000
0.799990000000000
0.799990000000000
0.776462799551987
0.759941339228894
0.744761469892433
0.730978756596800
0.718783678574067
0.708246462527232

Figure A.4: Screenshot of data S

Appendix B

Algorithms

B.1 CP3

```
function mat = CP3_algo(size,Tm,tolerance,VTX,N,S)
% size is the number of vectors will be used to generate the trajectory,
% Tm is the Tolerance method
% tol is the value of tolerance
% VTX, N, and S are the solutions to the PDEs

re_position(VTX);
% position center of vector field in the origin

tails = VTX;
% VTX contains the tails (bases) of vectors

subsample = generate_subsample(tails, size, num_spheres);
% num_spheres is a parameter used in the sampling

vector_heads = build_heads(tails,N);
% obtain the positions of heads of vectors in the vector field

tail_dict = construct_dictionary(tails_z, tails);
head_dict = construct_dictionary(heads_z, heads);
% in these two dictionaries, the key is the z values of tails/heads,
% and the items related to the key are all tails/heads with this z.

key_tail = keys(tail_dict);
% this returns all keys - different z values

for all i in 1:length(key_tail) :
    heads_and_tails(i) = {tails.cylinder, heads.cylinder,
                          tails.cartesian, heads.cartesian, states};
    sort(heads_and_tails(i), tails.cylinder.theta);
end
% state contains information if a vector (tail&head) is discovered or not.
% sort heads_and_tails by value of theta of tails

cyl_dict = construct_dictionary(key_tail,heads_and_tails);
% dictionary: key = tail.cartesian.z  value = vectors

reach_top = false;
end_curLayer = false;

cur_vector = heads_and_tails(1,1); % first vector in first layer

while reach_top == false
    if end_curLayer == false
        append(mat,cur_vector.tail.cartesian);
        append(mat,cur_vector.head.cartesian);
        % add current vector to output matrix

        cur_vector.state = true;
        % mark current vector as discovered

        layer = cyl_dict(cur_vector.tails.cartesian.z);
        % obtain the layer of current vector
```

```
        layer(find(cur_vector)) = cur_vector;
        cyl_dict(cur_vector.tails.cartesian.z) = layer;
        % update states in cyl_dict

        cur_z = cur_vector.head.cartesian.z;
        % find next vector using current head's height
    end

    if isKey(cyl_dict, cur_z) && end_curLayer == false
        key = cur_z;
    else
        key = find(min(key_tail>cur_z));
    end
    % look for next layer

    layer = cyl_dict(key);
    cur_head_dist_to_z_axis = sqrt(cur_vector.head.cartesian.x^2+
                                   cur_vector.head.cartesian.y^2);
    next_tails_dists_to_z_axis = sqrt(layer.tails.cartesian.x.^2+
                                      layer.tails.cartesian.^2);
    % compute the distances for tolerance restriction

    if Tm == 1
        if cur_head_dist_to_z_axis <= 0.1
           % radius of disordered inner core
            tol = tolerance/5;
        else
            tol = tolerance;
        end
    elseif Tm == 2
        tol = tolerance/(cur_head_dist_to_z_axis/0.5);
        % radius of the capsid is 0.5
    else
        report_error('invalid Tm type');
    end

    next_vectors = find(
           layer.tails.cylinder.theta > cur_vector.head.cylinder.theta &&
           layer.states == false &&
           next_tails_dists_to_z_axis < cur_head_dist_to_z_axis + tol &&
           next_tails_dists_to_z_axis > cur_head_dist_to_z_axis - tol);
    % find next tails

    if isempty(next_vectors) == false
        % there are multiple possible tails
        cur_vector = find(min(next_vectors.tail.cylinder.theta));
    elseif layer(1).state == false
        % current vector has largest theta, look for the smallest one
        cur_vector = layer(1);
    else
        % all vectors are discovered in this layer
        end_curLayer = true;
    end
end
```

B.2 CP2

```
function mat = CP2_algo(size,VTX,N,S)
re_position(VTX);
tails = VTX;
subsample = generate_subsample(tails, size, num_spheres);
vector_heads = build_heads(tails,N);
tail_dict = construct_dictionary(tails_z, tails);
head_dict = construct_dictionary(heads_z, heads);
key_tail = keys(tail_dict);
for all i in 1:length(key_tail) :
    heads_and_tails(i) = {tails.cylinder, heads.cylinder,
                          tails.cartesian, heads.cartesian, states};
    sort(heads_and_tails(i), tails.cylinder.theta);
end
cyl_dict = construct_dictionary(key_tail,heads_and_tails);
reach_top = false;
end_curLayer = false;
cur_vector = heads_and_tails(1,1);
while reach_top == false
    if end_curLayer == false
        append(mat,cur_vector.tail.cartesian);
        append(mat,cur_vector.head.cartesian);
        cur_vector.state = true;
        layer = cyl_dict(cur_vector.tails.cartesian.z);
        layer(find(cur_vector)) = cur_vector;
        cyl_dict(cur_vector.tails.cartesian.z) = layer;
        cur_z = cur_vector.head.cartesian.z;
    end
    if isKey(cyl_dict, cur_z) && end_curLayer == false
        key = cur_z;
    else
        key = find(min(key_tail>cur_z));
    end
    layer = cyl_dict(key);

%%%%%%%%%%%%%%%%%%%%%%% The above is same as CP3 %%%%%%%%%%%%%%%%%%%%%%%

    next_vectors = find(
           layer.tails.cylinder.theta > cur_vector.head.cylinder.theta &&
           layer.states == false);

%%%%%%%%%%%%%%%%%%%%%%% The below is same as CP3 %%%%%%%%%%%%%%%%%%%%%%%

    if isempty(next_vectors) == false
        cur_vector = find(min(next_vectors.tail.cylinder.theta));
    elseif layer(1).state == false
        cur_vector = layer(1);
    else
        end_curLayer = true;
    end
end

end
```

References

[1] Colin C Adams. *The knot book*. American Mathematical Soc., 1994.

[2] Javier Arsuaga and Y Diao. "DNA knotting in spooling like conformations in bacteriophages". In: *Computational and Mathematical Methods in Medicine* 9.3-4 (2008), pp. 303–316.

[3] Uri M Ascher and Chen Greif. *A first course on numerical methods*. SIAM, 2011.

[4] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[5] Richard H Crowell and Ralph Hartzler Fox. *Introduction to knot theory*. Vol. 57. Springer Science & Business Media, 2012.

[6] Irshad Ul Haq et al. "Bacteriophages and their implications on future biotechnology: a review". In: *Virology journal* 9.1 (2012), pp. 1–8.

[7] E Kellenberger et al. "Considerations on the condensation and the degree of compactness in non-eukaryotic DNA-containing plasmas". In: *Bacterial chromatin*. Springer, 1986, pp. 11–25.

[8] Amelie Leforestier and Francoise Livolant. "The bacteriophage genome undergoes a succession of intracapsid phase transitions upon DNA ejection". In: *Journal of molecular biology* 396.2 (2010), pp. 384–395.

[9] WB Raymond Lickorish. *An introduction to knot theory*. Vol. 175. Springer Science & Business Media, 2012.

[10] Françoise Livolant. "Ordered phases of DNA in vivo and in vitro". In: *Physica A: Statistical Mechanics and its Applications* 176.1 (1991), pp. 117–137.

[11] Davide Marenduzzo et al. "DNA–DNA interactions in bacteriophage capsids are responsible for the observed DNA knotting". In: *Proceedings of the National Academy of Sciences* 106.52 (2009), pp. 22269–22274.

[12] Igor Muševič. "Nematic liquid-crystal colloids". In: *Materials* 11.1 (2018), p. 24.

[13] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[14] Udom Sae-Ueng et al. "Solid-to-fluid DNA transition inside HSV-1 capsid close to the temperature of infection". In: *Nature chemical biology* 10.10 (2014), p. 861.

[15] Alexander Sulakvelidze, Zemphira Alavidze, and J Glenn Morris. "Bacteriophage therapy". In: *Antimicrobial agents and chemotherapy* 45.3 (2001), pp. 649–659.

[16] Inc. The MathWorks. *MATLAB Optimization Toolbox*. Natick, Massachusetts, United State, 2020. URL: `https://www.mathworks.com/help/optim/`.

[17] Shelly Tzlil et al. "Forces and pressures in DNA packaging and release from viral capsids". In: *Biophysical journal* 84.3 (2003), pp. 1616–1627.

[18] Shawn Walker et al. "Liquid crystal model of viral DNA encapsidation". In: *Phys. Rev. E* 101 (2020), p. 022703.

[19] Walker, Shawn and Arsuaga, Javier and Hiltner, Lindsey and Calderer, M Carme and Vázquez, Mariel. "Fine structure of viral dsDNA encapsidation". In: *Physical Review E* 101.2 (2020), p. 022703.