Robust implementation of algorithms to characterize topology in DNA

A Report on Laboratory Project

Submitted for the Partial Fulfilment of the

DEGREE OF

FIVE YEAR INTEGRATED MASTER OF SCIENCE

IN

CHEMISTRY

ANURAG SINGH

[12310005]

DEPARTMENT OF CHEMISTRY

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

ROORKEE-247667 (INDIA)

April, 2017

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE

CANDIDATE'S DECLARATION

I hereby certify that the work, which is being presented in this project report, entitled **Robust implementation of algorithms to characterize topology in DNA** in partial fulfillment of the requirements for the award of the Degree of Master of Science and submitted in the Department of Chemistry of the Indian Institute of Technology Roorkee is an authentic record of my own work carried out during a period 6 January 2017 to 20 April 2017 under the supervision of Dr Pallavi Debnath. The matter presented in the thesis has not been submitted by me for the award of any other degree of this or any other Institute.

Anurag Singh

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: _____ **Dr Pallavi Debnath**

Abstract

The progress in the study of DNA topology requires faster and effortless tools for topological characterization of DNA knots. In this project I develop a robust computer program to characterize knots by Alexander polynomial, a topological invariant. This study will provide a basis for further studies on the mechanism of enzyme action towards selective topology simplification in circular DNA.

Acknowledgments

I would like toexpress my deepest gratitude and sincere thanks to my project supervisor, Dr. Pallavi Debnath. She not just only guided and motivated me, but also supported and encourage the formulation of new ideas, which turned out to be one of the best academic experience so far during my stay at IIT Roorkee .This will be always a valuable experience for me.

I am also grateful our Head of Chemistry Department, Dr. Manner R. Maurya and our Faculty advisor, Dr. K. R. Justin Thomas, for providing this opportunity to get hands on experience of research work.

Last but not the least, I would also like to thank all the committee members for their constant support and assistance during this project. I would like to thank them for providing us with all the necessary facilities to carry this project.

Picture Table

Table of Tables

Table of Symbols

DNA	Deoxyribonucleic acid
dsDNA	Double Stranded DNA

Content

Introduction and Literature Survey

Topology defines the properties of space or structures that are preserved under continuous changes, such as stretching, crumpling and bending, but not tearing or gluing. Important topological properties include connectedness and compactness. The packing, twisting, and topological constraints considered altogether leads to topological entanglement, it can pose severe functional obstacles for study of DNA. It is developed as a field of study out of geometry and set theory, through analysis of concepts such as space, dimension, and transformation.

DNA play a crucial role in all living organisms because it is the key molecule responsible for storage duplication and realization of genetic information. DNA is a heteropolymeric molecule consisting of residue of 4 type A, T, C and G. [1]
High degree of uncertainty is found in eukaryotic DNA replication. Although the spatial and temporal pattern of DNA replication must complete the replication process in an accurate and timely manner. Failure to replicate even a small part of the genome would disrupt proper segregation of the genetic material to the two daughter cells during mitosis, leading to genomic instability.[1]

The two strands of DNA form right handed doubly helixes molecule. There are enzyme that are responsible for maintaining the geometry and topology by passing one strand of DNA through another by breaking of a transient enzyme-bridged break in one of the DNA strands, also another enzyme cut the DNA and recombine the cut ends by exchanging the move performed by above recombinase enzymes.

Topoisomerase

Topoisomerases are ubiquitous proteins found in all three domains of life (bacteria, archaea and eukarya). They are capable of changing the topology of DNA via transient

breaks on one or two of the DNA strands to allow passage of either a single or double DNA strand through the break.

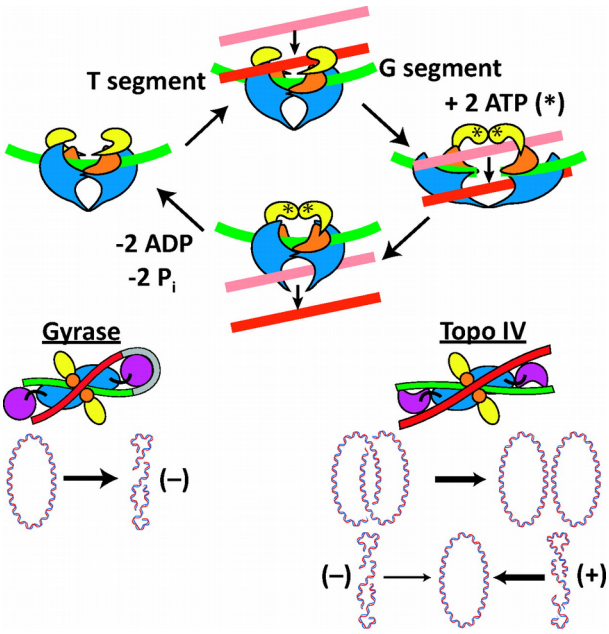


Figure 1 Topoisomerases

Topoisomerases have been involved in several cellular processes, such as transcription, replication and recombination, and the importance of their cellular role is underscored by the fact that they are the target of several cancer chemotherapeutic agents and antibiotics .Topoisomerases are classified into two types: type I enzymes cleave one DNA strand and pass either one or two strands through the break before resealing it, while type II molecules cleave both DNA strands in concert and pass another double strand through the break followed by religation of the double strand break. The reaction is takes the energy stored in the supercoiled DNA as torsional strain. No external source of energy is needed for the cleavage/religation part of the reaction Type II enzymes , but they do utilize ATP hydrolysis to drive conformational changes in the protein during the reaction cycle.[10][11]

These enzymes catalyse the process of altering the DNA topology by creating knots & links, producing DNA supercoiling and also sometimes by unlinking and unknotting.

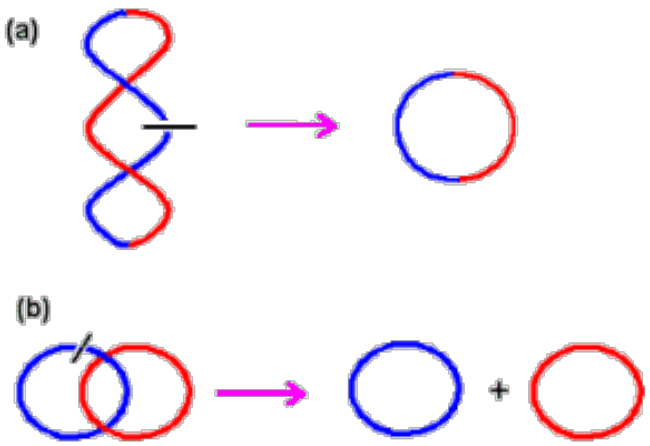


Figure 2. Example showing Topoisomerase II enzyme in mutant yeast

In order to understand the mechanisms of enzyme and their consequences on structure of DNA due to the action of enzymes, we need the serious knowledge of geometry and mathematical models to explain the DNA topology. The use of mathematical analytical theories becomes an important aspect to study topology. Topoisomerases are crucial for the survival of DNA. Despite, because of their ability cut the genetic material, the Topoisomerases also have the probability of fragmenting the genome. This character of topoisomerases is utilized in few of the most widely prescribed anticancer and antibacterial drugs presently used in the clinics. The concepts of topology are also applied to the genetic material. If the ends of DNA helix are fixed, the topological relations are characterized as those that can be modified only by cutting one or both strands of the double helix. The length of chromosomes, bacterial chromosomes, and plasmids are associated with a high frictional energy, therefore, they are often circular. Chromosomal DNA is bound to membranes in bacteria and scaffold in eukaryotes. Topological activities sighted in DNA are knotting, under-winding, over-winding, and tangling. DNA strand breaks by Topoisomerase can induce specific types of leukaemia. Topology is essentially affected in every nucleic acid process that demands the double helix to be broken or moved within the cell environment.[12]

Knots are closed loops which are tabulated and characterised by the crossing number. Crossing number is the number of times the curve of a knot has crossed itself given that the knot exists in its simplest form. The Alexander Briggs notation representation of knots is of form Nm .

N stands for the total number of crossing and "m" is the mth knot of a particular crossing number in a knot table which includes only primary knots. Many knots have been simplified by the mathematicians, the knots in simplest form is known as primary knots, the knots which are not in its simple configuration and can be further simplified by cutting through its strands of string and retying the ends giving nontrivial knots are known as composite knots. [9]

In the graphical form the knots are represented by crossing arc in which the crossing over arc is represented as continuous line and arc crossing under is represented as broken line.

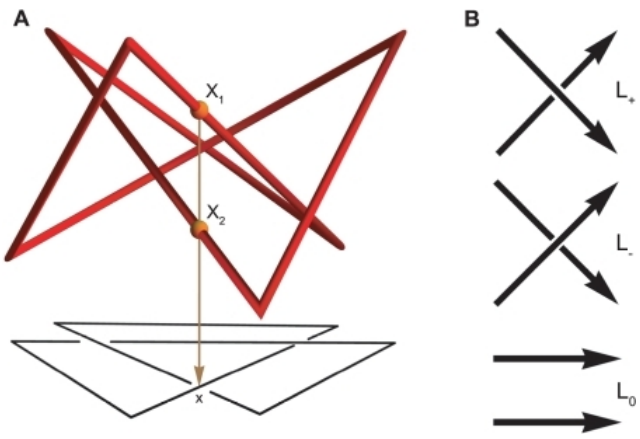


Figure 3. Knot Representation

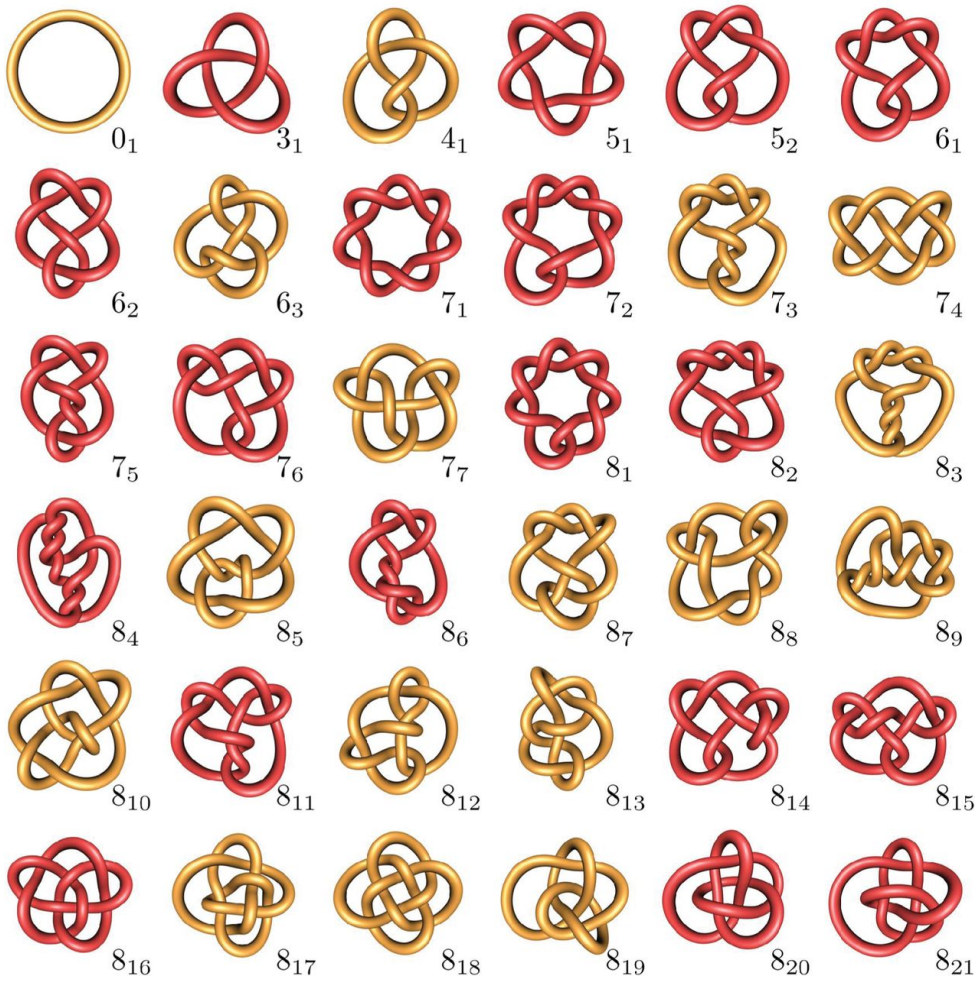


Figure 4. Primary knot diagrams till linking number eight.

Knots can form via two general mechanisms: threading of loose ends or breaking and re-joining of segments. Linear double-stranded DNA molecules undergoing random cyclization in solution exemplify the first mechanism. Cyclization is possible when the ends of a linear dsDNA molecule have complementary single-stranded overhangs. A knotted molecule results whenever the molecule's ends pass through loops within the same molecule

before joining .

Knots in ds DNA were first observed in 1980 when supercoiled plasmid was incubated with excess amounts of the type II topoisomerase from bacteriophage T4 (Liu, Liu, and Alberts 1980). DNA knotting can be manipulated both in vitro And in vivo and does occur naturally as well (Wasserman and Cozzarelli 1986; Wasserman and Cozzarelli 1991; Rybenkov, Cozzarelli, and Vologodskii 1993; Rodriguez-Campos 1996; Vologodskii 1999).

Reidemeister. Moves

German topologist Kurt Reidemeister observed that in few cases deformation of the knot does not result in alteration of knot types; these moves are called Reidemeister moves after the person who suggested it. Vice versa, it was concluded that whenever two-knot diagrams are equivalent, one-knot diagram can be converted to another knot diagram by applying the sequence of Reidemeister moves. Luckily, there are only three types of Reidemeister. Moves. Shown in the figure 5. Although, with the help Reidemeister moves we can directly know if two knots are equivalent but in the case of knots having a high linking number or large crossing points, applying Reidemeister move becomes difficult. Hence, this method is inefficient in those cases. An invariant is a property of an object which remains constant with respect to some transformation made under the Reidemeister moves. Which means that in the case of a knot if two knots are equivalent, they must have the same particular value of invariant.

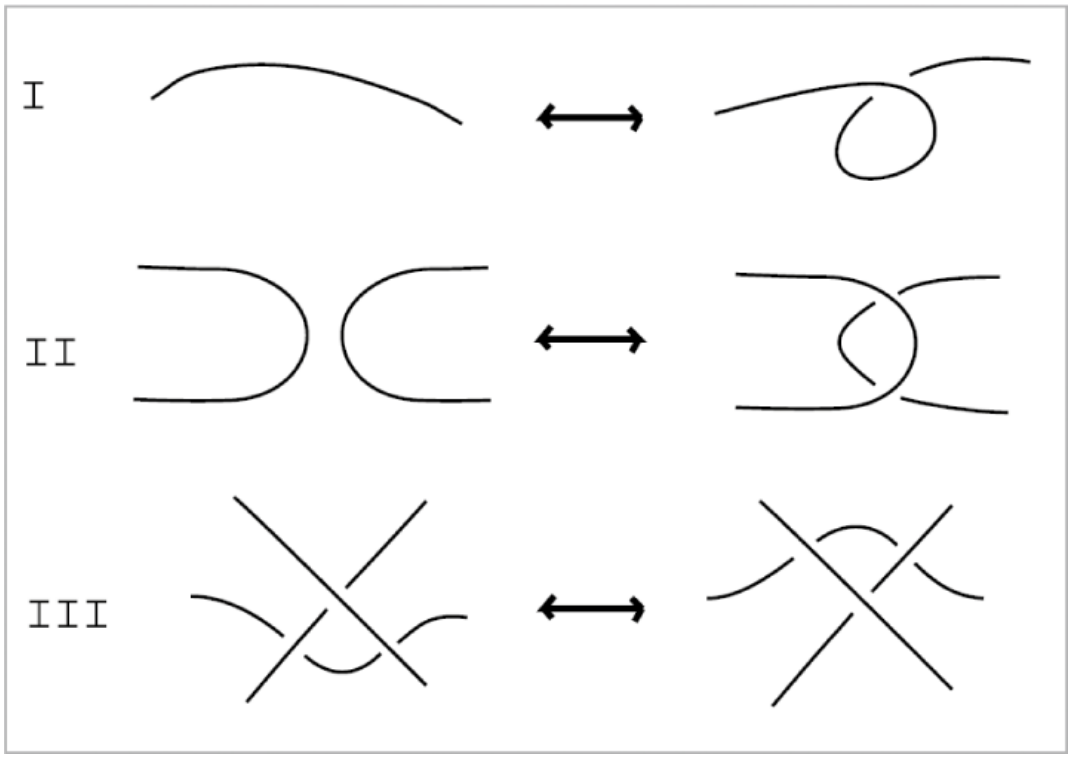


Figure 5 Reidemeister moves

Adams, Lickorish Rolfsen described knot invariant as a "quantity" that is the same for equivalent knots if the invariant is computed from a knot diagram, it should give the same value for two knot diagrams representing equivalent knots. An invariant may take the same value on two different knots, so by itself may be incapable of distinguishing all knots.

A knot polynomial is a knot invariant that is a polynomial. Well-known examples include the Jones and Alexander polynomials. Kauffman polynomial

Jones Polynomials was first discovered by Vaughan Frederick Randal Jones in 1984. It has been further modified and improved further. In further papers mathematical proofs and argument over its approach of its various part has been discussed. It is mostly used to analyze the knot topology of quantum mechanic field. Jones polynomial are being implemented in quantum bits using 4 quantum bit.[4][7]

Kauffman polynomial is a two variable knot polynomial whose theory was developed by Louis Kauffman it uses link diagram it is being used in string theory[5][8]

The above polynomial are very efficient in analysing the knot. But they are very complex and involve multiple steps on of Reidemeister moves and linking and de-linking. So it will use large computational power and time.

In this work algorithm to find Alexander polynomials is implemented. It is because Alexander polynomial has an important advantage that makes it very suitable for use in the statistical physics of ring chains — the simplest knots have almost unique Alexander polynomials. An invariant in the form of an Alexander polynomial also exists for linkages, but it is a function of two variables, rather than one (or more than two, depending on the number of chains being considered). There is a very simple relation between the Alexander polynomial and the Gauss integral of :

$$|G| = |\Delta(1.1)|$$

Hence we see that the Gauss integral is generally a far weaker invariant than the Alexander polynomial. An especially substantial defect of the Gauss integral as an invariant of a linkage is that it vanishes for many rather simple types of linkages and thus does not enable one to distinguish them from unlinked curves[11]

Alexander polynomials fails to distinguish between two knot when number of crossing increases. But other advance algorithm uses higher computational power. Also the probability of DNA knot of higher prime knot is less. So to increase computational efficiency we can use simple algorithms. Also There is no standardized software package available that perform such characterization. In all the works these characterization is either done manually by human which consume lots of time or their in house developed systems. That's also is the reason i have to start from scratch.

Computing the polynomial

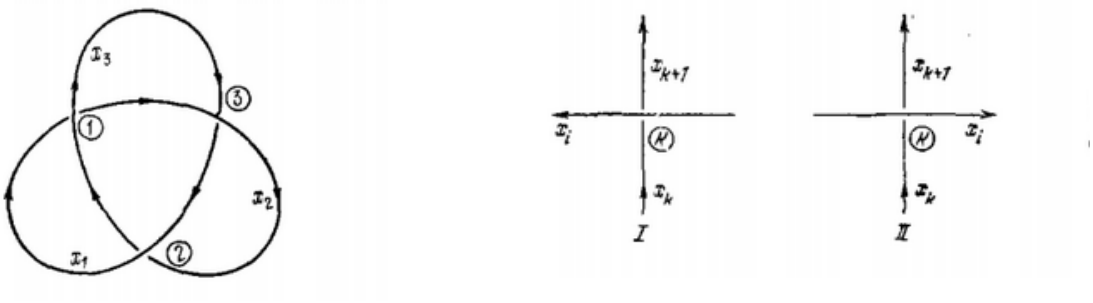


Figure 6 (a)Trefoil with named curve (b) Two types of crossing encountered

First the knot is projected on a plane along an arbitrarily chosen axis while drawing breaks at the crossing points in the part of the curve that lies below, generally the program will choose XY plane for the projection. Now the projection of the knot amounts to the set of segments of curves, which are called the generators. Arbitrary direction of looping of the generators was fixed and they were orderly indexed, having selected arbitrarily the first generator. The crossing that separates the k^{th} and $(k + l)^{th}$ generators will be identified by the k^{th} crossing. The crossings are of two types (Fig. 6(b)). Thus each crossing is characterized by its number, by its type (I or II), and by the number of the generator passing over it. Now we can generate Alexander matrix of the knot, in which the k^{th} row corresponds to the k^{th} crossing and which consists of n elements (n is the total number of crossings in the projection of the knot). Here all the elements except a_{kk}, $a_{k,k+1}$ and a_{ki} (i is the number of the overpassing generator) are zero. The nonzero elements of the k^{th} row are defined as follows:[13]

1) when i = k or i = k + 1 , independently of the type of crossing, we have:

$a_{kk} = -1$, $a_{k,k+1} = 1$

2) when i=k, i=k+1, we have for a type I crossing:a

$a_{kk} = -1$, $a_{kk+i} = -t$, $a_{ki} = t - 1$

and for a type II crossing:

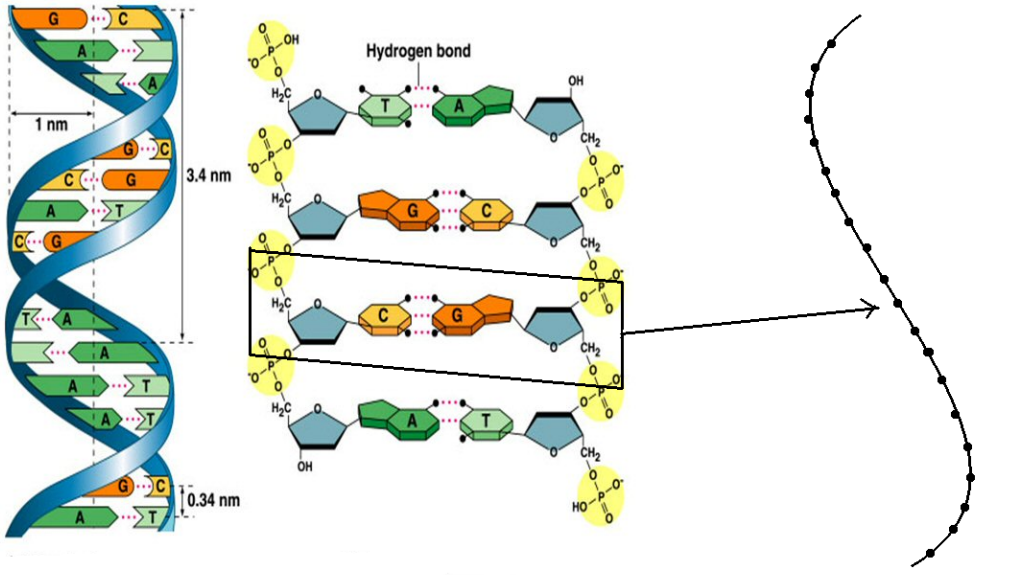
$a_{kk} = -t$, $a_{kk+i} = -1$, $a_{ki} = t - 1$

these relations ship holds for all k = 1……..n

In order to implement the algorithm the entire process was divided in two parts computation and analysis

In computational part a program was made to identify various parameter that are present in the given knot and then derive the corresponding matrix as per the algorithm of Alexander polynomial and the files were generated which was used for further analysis and visualization for better understanding.

The program is compiled code written in c++. It take a file as input that contain information of the 3D structure of the DNA. The 3D structure of DNA is helix coiled around a protein histone but to study the basic topology and for our analysis such a detailed structure was not needed, in fact it increases the compute time so for simplicity shake we can assume it to be a curve of zero width. See figure 7 This can be achieved by approximating the position of the base pair to be at their center of mass. Then this curve was used to for further characterization



(a) Key features of DNA structure (b) Partial chemical structure (c) Approximated structure

Figure 7. Approximating DNA

In C++ code file util.h file contains function to perform basic 3D vector calculations that includes 3D vector addition, subtractions, cross product and dot product , intersection of lines. And in the file main.cpp the advance stage of computation is coded. After reading the file that contain structural information of the DNA the obtained 3D curve is projected on a 2D plane . All the curve will be projected on XY plane unless or until any conflict arise. Conflicts could be the shapes in which relevant information is lost upon projecting it on XY plane. In those cases the result may be affected. After projecting it on a plane all the 3D coordinate of the knots is reduced to 2D coordinate, then the program search for all the crossings. To find the cross a very clever algorithm similar to Bentley–Ottmann algorithm is used. All the line segments forming the curve is sorted on the basis of their increasing value of minimum x coordinate of the line and also in increasing order of the maximum x coordinate of the line. So for any line to intersect with the given line segments must share common region . As show in the figure 8 suppose we have to find line segment crossing the line segment with the color maroon. To find that we may check if each and every line segment intersect the given

line. But it is visible from the figure that only a few whose some part lie in green region are more likely to intersect. And Thus we can reduce the number of line segments to be checked for intersection. That will make the process faster. To do that we just have to check if the maximum X of line segment(shown in red color dot in figure) is greater than the minimum X of the given line segment(shown in green color dot in figure) and minimum X of line segment is smaller than the maximum X of the given line segment. In this way we reduced the number of line segment to be checked for crossing from 16 to 5.

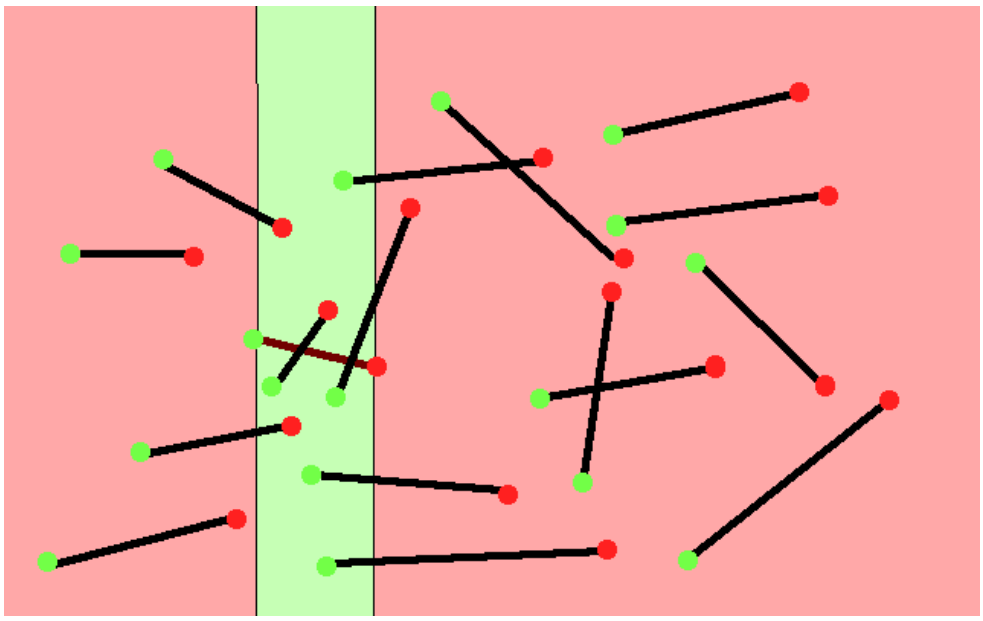


Figure 8 Bentley–Ottmann algorithm

In this way the number of tryout to find all the line segment intersecting a given line segment and the resultant intersection point was significantly reduced. The by doing similar operation with Y coordinates the operation of finding the crossing was more improved. After getting the intersection point we classified the line segments between the two crossing as an arc as shown in the figure. It is done by a loop in program from line 229, where it follow the path of knot and add line segments to array and program come to a crossing it program starts adding line segments to new array

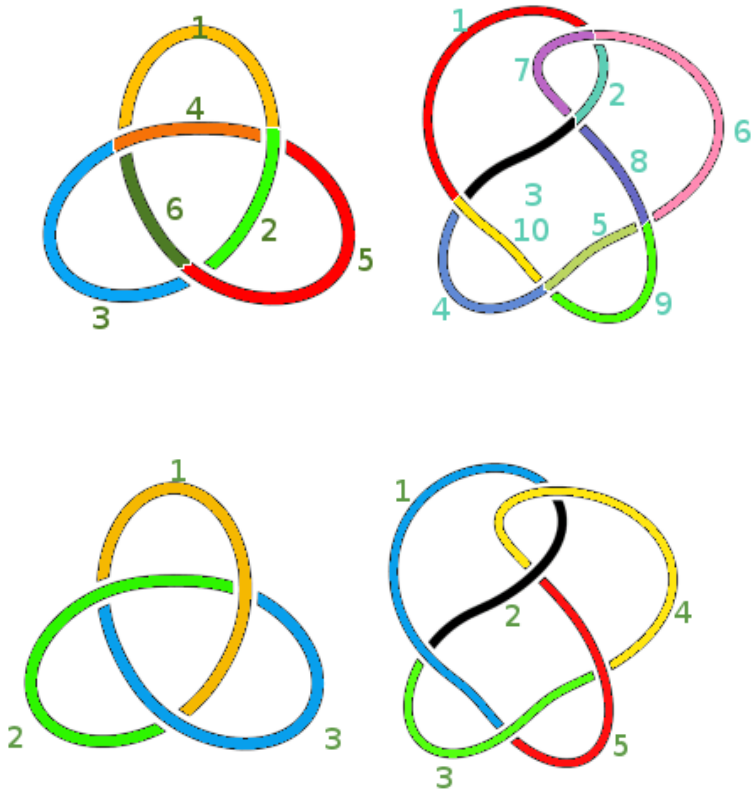


Fig 9(a) The arc joining any two crossing are named as separate arc coded with different color (b) The arc which are crossing over are joined together.

Then the arc that started from crossing over was joined to the arc previous to it. Thus we got arcs that starts from crossing under to crossing below. As we can see in the figure 9 (a) the all the arc joining any two crossing are marked as a separate arc and then in the figure 9(b) the two arc sharing a crossing over were joined together. So from the figure 9(a) the arc 1 and 2 were joined to make arc 1in figure 9(b) and so on. To identify which arc is crossing over and which is crossing under we check the z value of the crossing point lying on the arc. While joining arcs, the arc that was going over was also noted Then for each crossing by using index of the arc passing over, we created a matrix as

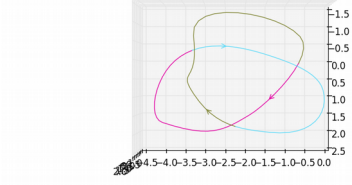
per the rules discussed above and by calculating the determinanat of the matrix we derived the invariant.

Analysis Script

Since the data that is given is very complex . so few tools to analyze the result programs in python using matplotlib graph plotting library were developed that helped to visualise the data that comes out of the program after each steps. It was used to check if the program performed well or not. In the visualization part it is explained thoroughly in next section.

Experiments and result

Starting task was to make program to that could read the coordinate from the file and search for the crossing in any shapes or polygon. To test the code if it is finding the crossings correctly. Few polygons were generated with random sides then given to the program as input. It identified the crossing correctly. The accuracy to calculate the crossing points coordinate was set till 5 places of decimal . It can be increased or decreased as per requirement of the given structure. Knot of 13 types were prepared on a 3D CAD software name blender which is an open source 3D as modeling software modeling software as per figure in paper[citation]. The program give out data which it the information about the arcs that was formed from collecting lines from joining the two crossings. Then after joining the crossing-over arcs with the arc proceeding it output about the description of the arc is written to file. A python script a4.py plots the data in 3D graph



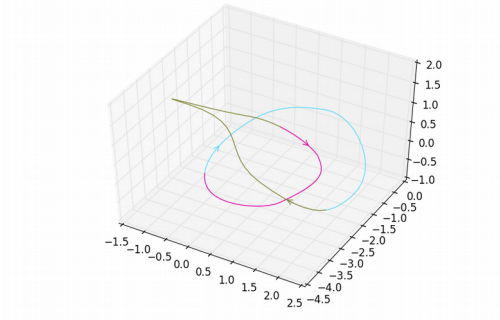


Figure 10(a) Top view of 3_1 knot . Figure 10(b) 3_1 knot with elevation 45^o.

As show in the figure 10(a) and 10(b) the resultant arc was color coded with separate color and the direction which it is following the path is shown by the arrow. Form these data the matrix is derived for all the tested see appendix 2. By calculating the determinant polynomial is derived. It is shown in the table:

Type of knot	Alexander polynomial
3_1	t^2-t+1
4_1	t^2-3t+1
5_1	$t^4-t^3+t^2-t+1$
5_2	$2t^2-3t+2$
6_1	$2t^2-5t+2$
6_2	$t^4-3t^3+3t^2-3t+1$
6_3	$t^4-3t^3+5t^2-3t+1$
7_1	$t^6-t^5+t^4-t^3+t^2-t+1$
7_2	$3t^2-5t+3$
7_3	$2t^4-3t^3+3t^2-3t+2$
7_4	$t^4-5t^3+7t^2-5t+1$
7_5	$2t^2-4t^3+5t^2-4t+2$
7_6	$t^4-5t^3+7t^2-5t+1$
7_7	$t^4-5t^3+9t^2-5t+1$

Conclusion

The process has tried to be automated in every level from reading the data analysing data and visualizing data. The 13 knots has been tried out and the result was fine. This project delivers a robust system to identify knots. With tis system the biophysical analysis of DNA topology will be accelerated. Automation work has to further extended to for calculating other topological properties. This program will fail to identify higher prime knots as the Alexander Polynomial fail. So a hybrid method can be developed in which for identification lower prime knots Alexander polynomial will be used and for higher prime knots better algorithm such Jones and Kauffman polynomial can be used. This will computationally optimist the an will give higher performance.

Reference

1. Frank-Kamenetskii, Maxim D. "Biophysics of the DNA molecule." *Physics Reports*, vol. 288, no. 1-6, 1997, pp. 13–60.

2. Meluzzi, Dario, et al. "Biophysics of Knotting." *Annual Review of Biophysics*, vol. 39, no. 1, 2010, pp. 349–366.

3. Lygeros, J., et al. "Stochastic hybrid modeling of DNA replication across a complete genome." *Proceedings of the National Academy of Sciences*, vol. 105, no. 34, 2008, pp. 12295–12300.,

4. Jones, Vaughan F. R. "A polynomial invariant for knots via von Neumann algebras." *Bulletin of the American Mathematical Society*, vol. 12, no. 1, Jan. 1985, pp. 103–112.

5. Vahrenhold, Jan. "Line-Segment intersection made in-Place." *Computational Geometry*, vol. 38, no. 3, 2007, pp. 213–230.

6. Kauffman, Louis H. "Topological quantum information, Khovanov homology and the Jones polynomial." *Topology of Algebraic Varieties and Singularities Contemporary Mathematics*, 2011, pp. 245–264.

7. Aharonov, Dorit, et al. "A Polynomial Quantum Algorithm for Approximating the Jones Polynomial." *Algorithmica*, vol. 55, no. 3, 2008, pp. 395–421.

8. Aganagic, Mina. "String theory and math: Why this marriage may last. Mathematics and dualities of quantum physics." *Bulletin of the American Mathematical Society*, vol. 53, no. 1, 2015, pp. 93–115.

9. Vahrenhold, Jan. “Line-Segment intersection made in-Place.” *Computational Geometry*, vol. 38, no. 3, 2007, pp. 213–230.

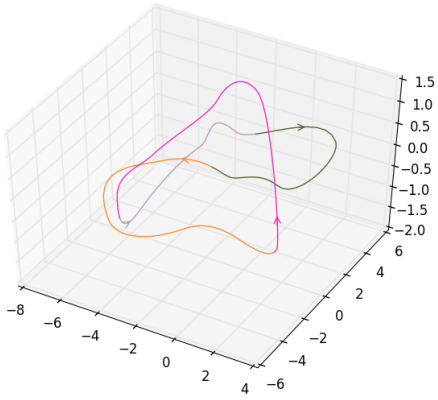
10. Schoeffler, A. J; Berger, J. M. Recent advances in understanding structure-function relationships in the type II topoisomerase mechanism. Biochem. Soc. Trans. 2005, 33, 1465–1470.

11. Vologodskii, A. Theoretical models of DNA topology simplification by type IIA DNA topoisomerases. Nucleic Acid Res. 2009, 37, 3125-3133.

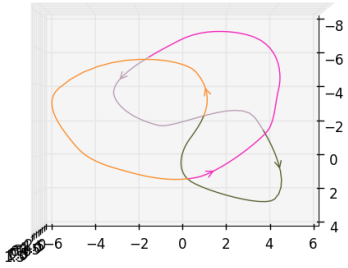
12. Liu, Di, et al. “Creating complex molecular topologies by configuring DNA four-Way junctions.” *Nature Chemistry*, vol. 8, no. 10, Apr. 2016, pp. 907–914.

13. Frank-Kamenetskiĭ, M D, and A V Vologodskiĭ. “Topological aspects of the physics of polymers: The theory and its biophysical applications.” *Soviet Physics Uspekhi*, vol. 24, no. 8, 1981, pp. 679–696.

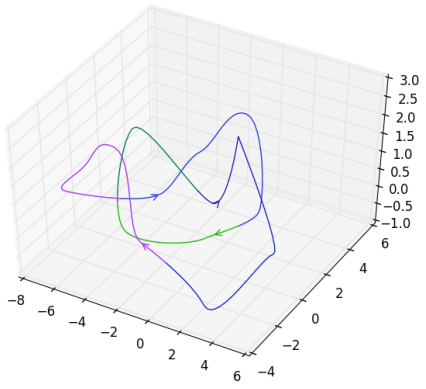
Appendix 1



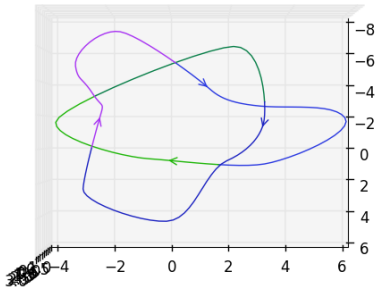
4_1 knot ,elevation 45°



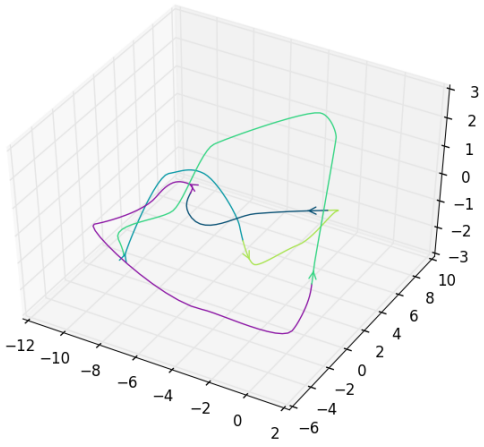
4_1 knot ,elevation 90°



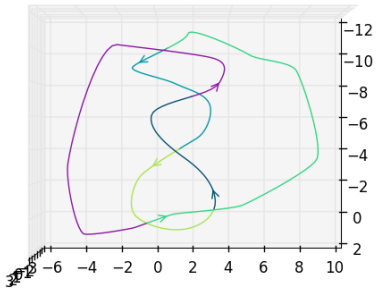
5_1 knot elevation 45°



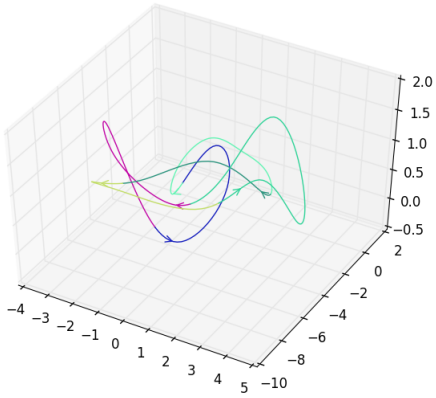
5_2 knot elevation 90°



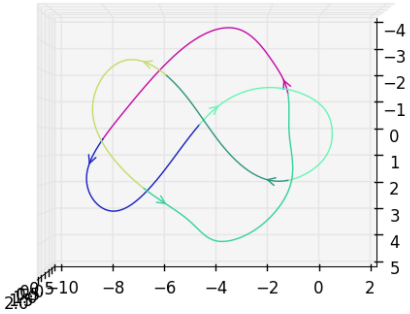
5_2 knot elevation 45°



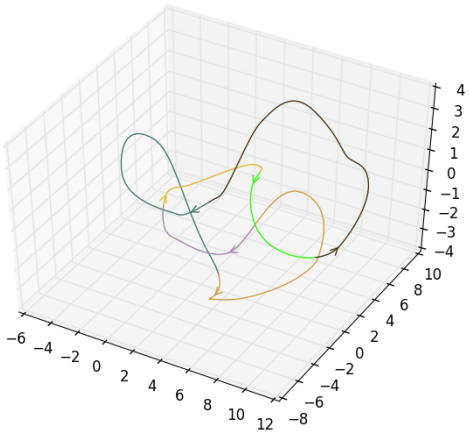
5_2 knot elevation 90°



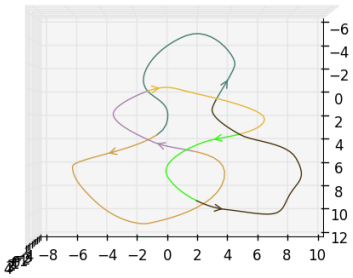
6_1 knot elevation 45°



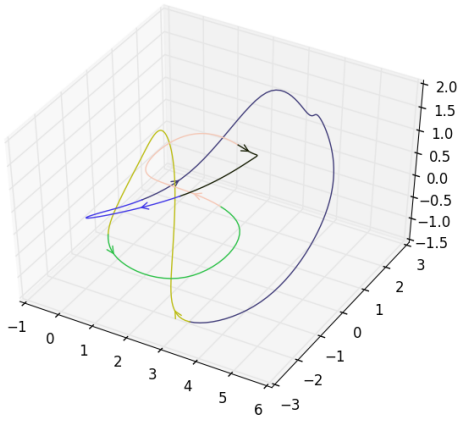
6_1 knot, elevation 90°



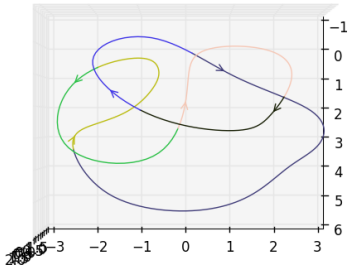
6_2 knot elevation 90º



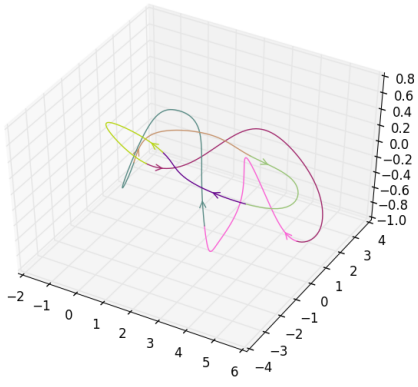
6_2 knot, elevation 90º



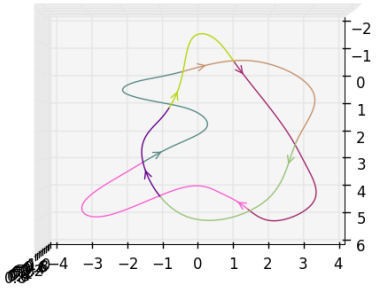
6_3 knot elevation 45°



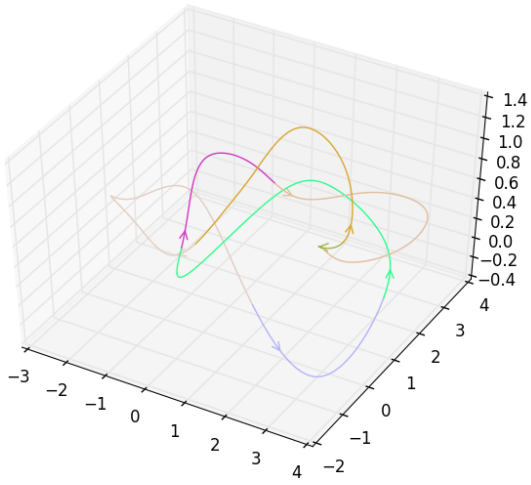
6_3 knot elevation 90°



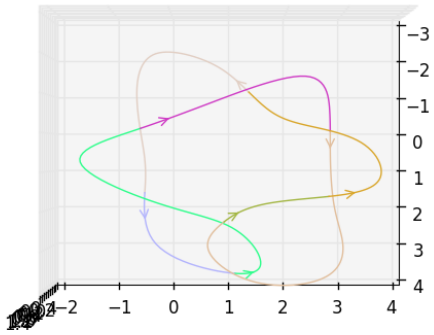
7_1 knot 45°



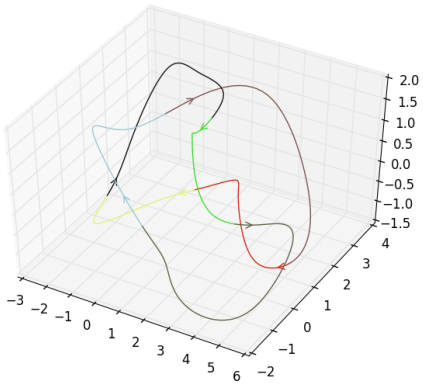
7_1 knot elevation 90°



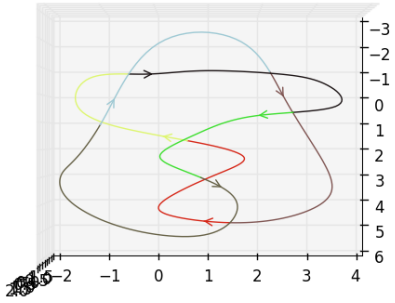
7_2 knot elevation 45°



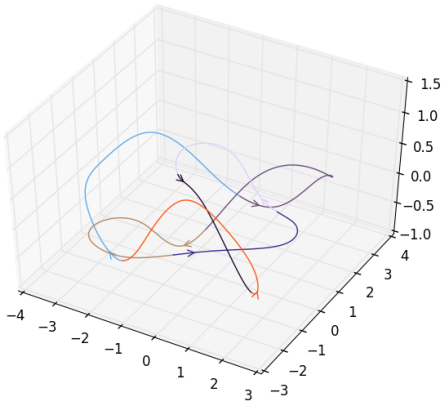
7_2 knot elevation 90°



7_3 knot elevation 45°

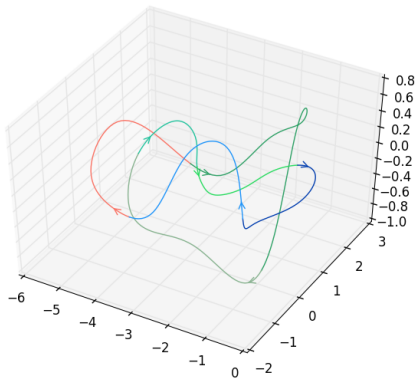


7_3 Knot elevation 90°

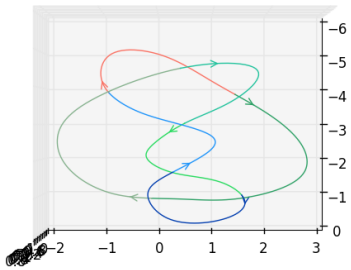


7_4 knot elevation 45°

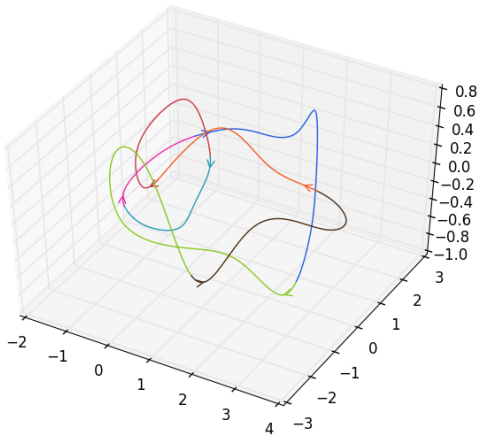
7_4 knot elevation 90°



7_5 knot elevation 45º

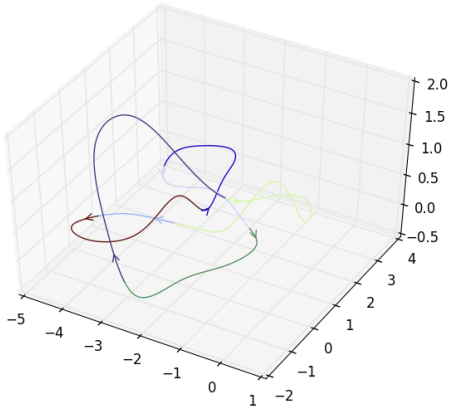


7_5 knot elevation 90º

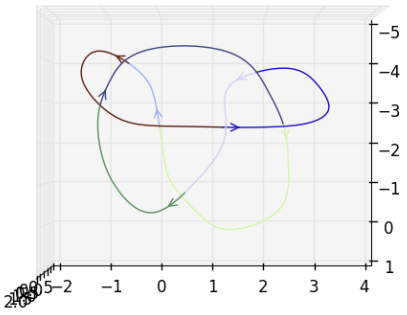


7_6 knot elevation 45°

7_6 knot elevation 90°



7_7knot elevation 45o



7_7 knot elevation 90^o

Appendix 2

Knot Matrix

$$\begin{bmatrix} -t & t-1 & 0 \\ 1 & -t & t-1 \\ t-1 & 1 & -t \end{bmatrix}$$

For 3_1 Knot

$$\begin{bmatrix} 1 & 0 & t-1 & 0 \\ 1 & -t & 0 & t-1 \\ t-1 & -t & 1 & 0 \\ 0 & t-1 & 1 & -t \end{bmatrix}$$

For 4_1 Knot

$$\begin{bmatrix} -t & 0 & t-1 & 0 & 0 \\ 1 & -t & 0 & t-1 & 0 \\ 0 & 1 & -t & 0 & t-1 \\ t-1 & 0 & 1 & -t & 0 \\ 0 & t-1 & 0 & 1 & -t \end{bmatrix}$$

For 5_1 Knot

$$\begin{bmatrix} 1 & 0 & 0 & t-1 & 0 \\ -t & 1 & t-1 & 0 & 0 \\ 0 & -t & 1 & 0 & t-1 \\ t-1 & 0 & -t & 1 & 0 \\ 0 & t-1 & 0 & -t & 1 \end{bmatrix}$$

For 5_2 Knot

$$\begin{bmatrix} 1 & 0 & t-1 & 0 & 0 & 0 \\ 1 & -t & 0 & 0 & t-1 & 0 \\ 0 & 1 & -t & t-1 & 0 & 0 \\ 0 & 0 & -t & 1 & 0 & t-1 \\ t-1 & 0 & 0 & 1 & -t & 0 \\ 0 & t-1 & 0 & 0 & 1 & -t \end{bmatrix}$$

For 6_1 Knot

$$\begin{bmatrix} 1 & 0 & 0 & t-1 & 0 & 0 \\ -t & 1 & t-1 & 0 & 0 & 0 \\ 0 & 1 & -t & 0 & t-1 & 0 \\ t-1 & 0 & -t & 1 & 0 & 0 \\ 0 & 0 & 0 & -t & 1 & t-1 \\ 0 & t-1 & 0 & 0 & 1 & -t \end{bmatrix}$$

For 6_2 Knot

$$\begin{bmatrix} -t & 0 & 0 & t-1 & 0 & 0 \\ -t & 1 & t-1 & 0 & 0 & 0 \\ 0 & -t & 1 & 0 & 0 & t-1 \\ 0 & 0 & 1 & -t & t-1 & 0 \\ t-1 & 0 & 0 & 1 & -t & 0 \\ 0 & t-1 & 0 & 0 & -t & 1 \end{bmatrix}$$

For 6_3 Knot

$$\begin{bmatrix} -t & 0 & 0 & t-1 & 0 & 0 & 0 \\ 1 & -t & 0 & 0 & t-1 & 0 & 0 \\ 0 & 1 & -t & 0 & 0 & t-1 & 0 \\ 0 & 0 & 1 & -t & 0 & 0 & t-1 \\ t-1 & 0 & 0 & 1 & -t & 0 & 0 \\ 0 & t-1 & 0 & 0 & 1 & -t & 0 \\ 0 & 0 & t-1 & 0 & 0 & 1 & -t \end{bmatrix}$$

For 7_1 Knot

$$\begin{bmatrix} 1 & 0 & 0 & t-1 & 0 & 0 & 0 \\ -t & 1 & t-1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -t & 0 & t-1 & 0 & 0 \\ t-1 & 0 & -t & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -t & 1 & 0 & t-1 \\ 0 & 0 & t-1 & 0 & 1 & -t & 0 \\ 0 & 0 & 0 & 0 & t-1 & -t & 1 \end{bmatrix}$$

For 7_2 Knot

$$\begin{bmatrix} -t & 0 & 0 & t-1 & 0 & 0 & 0 \\ 1 & -t & 0 & 0 & 0 & t-1 & 0 \\ 0 & 1 & -t & 0 & t-1 & 0 & 0 \\ 0 & 0 & 1 & -t & 0 & 0 & t-1 \\ t-1 & 0 & 0 & 1 & -t & 0 & 0 \\ 0 & t-1 & 0 & 0 & 1 & -t & 0 \\ 0 & 0 & t-1 & 0 & 0 & 1 & -t \end{bmatrix}$$

For 7_3 Knot

$$\begin{bmatrix} -t & 0 & t-1 & 0 & 0 & 0 & 0 \\ 1 & -t & 0 & 0 & t-1 & 0 & 0 \\ 0 & 1 & -t & 0 & 0 & t-1 & 0 \\ 0 & 0 & 1 & -t & 0 & 0 & t-1 \\ 0 & t-1 & 0 & 1 & -t & 0 & 0 \\ t-1 & 0 & 0 & 0 & 1 & -t & 0 \\ 0 & 0 & 0 & t-1 & 0 & 1 & -t \end{bmatrix}$$

For 7_4 Knot

$$\begin{bmatrix} 1 & 0 & t-1 & 0 & 0 & 0 & 0 \\ -t & 1 & 0 & t-1 & 0 & 0 & 0 \\ 0 & -t & 1 & 0 & 0 & t-1 & 0 \\ t-1 & 0 & -t & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -t & 1 & 0 & t-1 \\ 0 & t-1 & 0 & 0 & -t & 1 & 0 \\ 0 & 0 & 0 & 0 & t-1 & -t & 1 \end{bmatrix}$$

For 7_5 Knot

$$\begin{bmatrix} 1 & 0 & t-1 & 0 & 0 & 0 & 0 \\ -t & 1 & 0 & 0 & 0 & 0 & t-1 \\ 0 & -t & 1 & 0 & t-1 & 0 & 0 \\ t-1 & 0 & -t & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -t & t-1 & 0 \\ 0 & t-1 & 0 & 0 & -t & 1 & 0 \\ 0 & 0 & 0 & t-1 & 0 & 1 & -t \end{bmatrix}$$

For 7_6 Knot

$$\begin{bmatrix} -t & 0 & 0 & t-1 & 0 & 0 & 0 \\ -t & 1 & t-1 & 0 & 0 & 0 & 0 \\ 0 & -t & 1 & 0 & 0 & 0 & t-1 \\ 0 & 0 & -t & 1 & t-1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -t & 0 & t-1 \\ t-1 & 0 & 0 & 0 & -t & 1 & 0 \\ 0 & 0 & t-1 & 0 & 0 & -t & 1 \end{bmatrix}$$

For 7_7 Knot

Appendix 3

util.h

```
#ifndef UTIL_H

#define UTIL_H
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <vector>
#include <fstream>
#include <exception>
#include <string>
using namespace std;
#define PI 3.14159265359
long MAX_LIN = 0;
long MAX_PNT = 0;
class line;
typedef vector<line*> arc;
class pointer{
public:
        vector<double> k;
        vector<line*> parent;
        pointer():k(3),parent(0){
        }
        pointer(double a, double c): k(3), parent(0){
                k[0] = a;
                k[1] = c;
                k[2] = 0;
        }
        pointer(double a, double b,double c): k(3), parent(0){
                k[0] = a;
                k[1] = b;
                k[2] = c;
        }
        void operator=(const pointer &p){
                this->k[0] = p.k[0];
                this->k[1] = p.k[1];
                this->parent = p.parent;
        }
        pointer operator-(const pointer &p){
                pointer ret;
                ret.k[0] = this->k[0] - p.k[0];
                ret.k[1] = this->k[1] - p.k[1];
                ret.k[2] = this->k[2] - p.k[2];
                return ret;
        }

        pointer operator+(const pointer &p){
                pointer ret;
                ret.k[0] = this->k[0] + p.k[0];
                ret.k[1] = this->k[1] + p.k[1];
                ret.k[2] = this->k[2] + p.k[2];
                return ret;
        }

        pointer operator*(const double &p){
                pointer ret;
                ret.k[0] = this->k[0] *p;
```

```
                ret.k[1] = this->k[1] *p;
                ret.k[2] = this->k[2] *p;
                return ret;
        }

        double abs(){
                return sqrt(this->k[0]*this->k[0] + this->k[1]*this->k[1] + this-
>k[2]*this->k[2]);
        }

        pointer unit(){
                pointer l;
                double va = this->abs();
                l.k[0] = this->k[0]/va;
                l.k[1] = this->k[1]/va;
                l.k[2] = this->k[2]/va;
                return l;
        }
};
class arc_info{
public:
        int indx;
        arc* head;
        arc_info* back;
        void link( arc_info* head){
                if(head != NULL){
                        this->head = head->head;
                        this->headindx = head->headindx;
                        head->back = this;
                }
                if(this->back != NULL){
                        this->back->link(this);
                }
        }
        int headindx;
        arc_info(){
                head = NULL;
                back = NULL;
        }
};

class intersect_info{
public:
        arc* up;
        arc* down;
        int indx_up, indx_down;
        pointer* point;
        intersect_info(){
                up = NULL;
                down = NULL;
                point = NULL;
        }
};
class line {
public:
        vector<pointer*> l;
        pointer* arc_p1;
        pointer* arc_p2;
        vector<pointer*> IntersectionPoints;
        line():l(2), IntersectionPoints(0){
                arc_p1 = l[0];
```

```
                arc_p2 = l[1];
        }
        line(pointer* a, pointer* b): l(2), IntersectionPoints(0){
                l[0] = a;
                l[1] = b;
                arc_p1 = l[0];
                arc_p2 = l[1];
        }
        void operator=(const line& p){
                this->l[0] = p.l[0];
                this->l[1] = p.l[1];
        }
        double min_x(){
                if(l[0]->k[0] < l[1]->k[0]){
                        return l[0]->k[0];
                }
                else{
                        return l[1]->k[0];
                }
        }
        double max_x(){
                if(l[0]->k[0] > l[1]->k[0]){
                        return l[0]->k[0];
                }
                else{
                        return l[1]->k[0];
                }
        }
        double min_y(){
                if(l[0]->k[1] < l[1]->k[1]){
                        return l[0]->k[1];
                }
                else{
                        return l[1]->k[1];
                }
        }
        double max_y(){
                if(l[0]->k[1] > l[1]->k[1]){
                        return l[0]->k[1];
                }
                else{
                        return l[1]->k[1];
                }
        }
};

double max( double a, double b){
        if(a > b){
                return a;
        }
        else{
                return b;
        }
}

pointer* line_intersection_with(line lin, double n){
        //cout<<n<<" "<<lin.l[0].k[0]<<" "<<lin.l[0].k[1]<<" "<<lin.l[1].k[0]<<"
"<<lin.l[1].k[1]<<endl;
        if((n >= min(lin.l[0]->k[0], lin.l[1]->k[0]) ) && (n <= max(lin.l[0]->k[0],
lin.l[1]->k[0]))) {
                double k= (n-lin.l[0]->k[0])/(lin.l[1]->k[0] - lin.l[0]->k[0]);
```

```
                //cout<<k<<" "<<"file\n";
                if((k <= 1)&&(k >= 0)){
                        return new pointer(n,lin.l[0]->k[1]+ (lin.l[1]->k[1] - lin.l[0]-
>k[1])*k);
                }
                else{
                        throw exception();
                }
        }
        else {
                throw exception();
        }
}

pointer* lineIntersection( line lin, line lin2){
        double k1 = (lin.l[0]->k[0] - lin.l[1]->k[0]);
        double k2 = (lin2.l[0]->k[0] - lin2.l[1]->k[0]);
        if( k1 == 0 || k2 == 0){
                double m1 = k1/(lin.l[0]->k[1] - lin.l[1]->k[1]);
                double m2 = k2/(lin2.l[0]->k[1] - lin2.l[1]->k[1]);
                double c1 = lin.l[0]->k[0] - lin.l[0]->k[1]*m1;
                double c2 = lin2.l[0]->k[0] - lin2.l[0]->k[1]*m2;
                if(m1 == m2)
                        return NULL;
                double x = (c2 - c1)/(m1 - m2);

                double y =  m1*x + c1;
                return new pointer(y,x);
        }
        else{
                double m1 = (lin.l[0]->k[1] - lin.l[1]->k[1])/k1;
                double m2 = (lin2.l[0]->k[1] - lin2.l[1]->k[1])/k2;
                double c1 = lin.l[0]->k[1] - lin.l[0]->k[0]*m1;
                double c2 = lin2.l[0]->k[1] - lin2.l[0]->k[0]*m2;
                if(m1 == m2)
                        return NULL;
                double x = (c2 - c1)/(m1 - m2);

                double y =  m1*x + c1;
                return new pointer(x,y);
        }
}

pointer* Z_val( line lin, pointer p){
        double base = lin.arc_p1->k[0] - lin.arc_p2->k[0];
        if(base != 0){
                double ratio = (lin.arc_p1->k[0] - p.k[0])/ base;
                double z = lin.arc_p1->k[2] - ratio*(lin.arc_p1->k[2] - lin.arc_p2-
>k[2]);
                pointer* k  = new pointer(p.k[0], p.k[1], z);
                return k;
        }
        else{
                base = lin.arc_p1->k[1] - lin.arc_p2->k[1];
                if(base != 0){
                        double ratio = (lin.arc_p1->k[1] - p.k[1])/ base;
                        double z = lin.arc_p1->k[2] - ratio*(lin.arc_p1->k[2] -
lin.arc_p2->k[2]);
                        pointer* k  = new pointer(p.k[0], p.k[1], z);
                        return k;
                }
                else{
```

```
                        return NULL;
                }
        }
}

int partition_line(vector<line*> &A, vector<int> &B, int start, int end){

        int i = start +1;
        line* piv = A[B[start]];
        for(int j =  start + 1; j <= end; j++){
                if(A[B[j]]->min_x() < piv->min_x()){
                        swap(B[i], B[j]);
                        //cout<<j<<endl;
                        i+= 1;
                }
        }
        swap(B[start] , B[i -1]);
        return i -1;
}

void quick_sort_line(vector<line*> &A, vector<int> &B, int start, int end){
        if(start < end){
                int piv_pos = partition_line(A,B,start, end);
                quick_sort_line(A,B,start,piv_pos - 1);
                quick_sort_line(A,B,piv_pos + 1, end);
        }
}

int partition_lineS(vector<line*> &A, int start, int end){

        int i = start +1;
        line* piv = A[start];
        for(int j =  start + 1; j <= end; j++){
                if(A[j]->min_x() < piv->min_x()){
                        swap(A[i], A[j]);
                        //cout<<j<<endl;
                        i+= 1;
                }
        }
        swap(A[start] , A[i -1]);
        return i -1;
}

void quick_sort_lineS(vector<line*> &A,int start, int end){
        if(start < end){
                int piv_pos = partition_lineS(A,start, end);
                quick_sort_lineS(A,start,piv_pos - 1);
                quick_sort_lineS(A,piv_pos + 1, end);
        }
}

int partition_lineSminy(vector<line*> &A, int start, int end){

        int i = start +1;
        line* piv = A[start];
        for(int j =  start + 1; j <= end; j++){
                if(A[j]->min_y() < piv->min_y()){
                        swap(A[i], A[j]);
                        //cout<<j<<endl;
                        i+= 1;
                }
        }
        swap(A[start] , A[i -1]);
```

```
        return i -1;
}

void quick_sort_lineSminy(vector<line*> &A,int start, int end){
        if(start < end){
                int piv_pos = partition_lineSminy(A,start, end);
                quick_sort_lineSminy(A,start,piv_pos - 1);
                quick_sort_lineSminy(A,piv_pos + 1, end);
        }
}

int partition_lineSmaxy(vector<line*> &A, int start, int end){

        int i = start +1;
        line* piv = A[start];
        for(int j =  start + 1; j <= end; j++){
                if(A[j]->max_y() < piv->max_y()){
                        swap(A[i], A[j]);
                        //cout<<j<<endl;
                        i+= 1;
                }
        }
        swap(A[start] , A[i -1]);
        return i -1;
}

void quick_sort_lineSmaxy(vector<line*> &A,int start, int end){
        if(start < end){
                int piv_pos = partition_lineSmaxy(A,start, end);
                quick_sort_lineSmaxy(A,start,piv_pos - 1);
                quick_sort_lineSmaxy(A,piv_pos + 1, end);
        }
}

void read(char* argv, vector<pointer*> &Points, vector<line*> &lineset){
        ifstream file(argv);
        string line2;
        int pointCounter = 0, linesetCounter = 0;
        for(int i = 0; i < 1000; i++){
                if(!getline(file,line2)){
                        break;
                }
                const char* s = line2.c_str();
                double k, m, n;
                if(s == NULL){
                        break;
                }
                if(s[0] == '#'){};
                if(s[0] == 'o'){};
                if(s[0] == 'v'){
                        sscanf(s+ 1,"%lf %lf %lf",&m,&n,&k);
                        Points[pointCounter] = new pointer(m,n,k);
                        pointCounter++;
                }
                if(s[0] == 'l'){
                        int a, b;
                        sscanf(s+ 1,"%d %d",&a,&b);
                        line* li1 = new line((Points[a - 1]), (Points[b -1 ]));
                        Points[a-1]->parent.insert(Points[a-1]->parent.end() ,li1);
                        Points[b-1]->parent.insert(Points[b-1]->parent.end() ,li1);
                        lineset[linesetCounter] = li1;
                        linesetCounter++;
```

```
                }
                //printf("%d %s\n",i, s);
        }
        if(Points.size()> pointCounter){
                Points.erase(Points.begin() + pointCounter, Points.end());
        }
        if( lineset.size() > linesetCounter){
                lineset.erase(lineset.begin() + linesetCounter, lineset.end());
        }
}

int binarySearch_min_x(vector<line*> a,int l,int h, double key)
{
        //string fl ="binarySearch_min_x : ";
        //cout<<fl<<l<<endl;
        while(l<=h)
        {
                //cout<<fl<<l<<" "<<h<<endl;
                if(key <= a[l]->min_x()){
                        break;
                }
                if(key >= a[a.size() - 1]->min_x()){
                        return h;
                }
                int m = (h + l)/2;
                //cout<<key<<" "<<a[m]->min_x()<<" "<<a[m + 1]->min_x()<<" ";
                if((key >= a[m]->min_x())&&(key < a[m + 1]->min_x())){
                        return m;
                }
                else if((key <= a[m]->min_x())&&(key > a[m -1]->min_x())) {
                        return m ;
                }
                else if(key >= a[m + 1]->min_x() ) {
                        l = m;
                }
                else {
                        h = m;
                }
        }
        return -1;                //key not found
 }

int binarySearch_min_y(vector<line*> a,int l,int h,double key)
{
        //string fl ="binarySearch_min_x : ";
        //cout<<fl<<l<<endl;
        while(l<=h)
        {
                //cout<<fl<<l<<" "<<h<<endl;
                if(key <= a[l]->min_y()){
                        break;
                }
                if(key >= a[a.size() - 1]->min_y()){
                        return h;
                }
                int m = (h + l)/2;
                if((key >= a[m]->min_y())&&(key < a[m + 1]->min_y())){
                        return m;
                }
                else if((key <= a[m]->min_y())&&(key > a[m -1]->min_y())) {
                        return m ;
```

```
                }
                else if(key >= a[m + 1]->min_y() ) {
                        l = m;
                }
                else {
                        h = m;
                }
        }
        return -1;                //key not found
 }

int binarySearch_max_y(vector<line*> a,int l,int h,double key)
{
        //string fl ="binarySearch_min_x : ";
        //cout<<fl<<l<<endl;
        while(l<=h)
        {
                //cout<<fl<<l<<" "<<h<<endl;
                if(key <= a[l]->max_y()){
                        break;
                }
                if(key >= a[a.size() - 1]->max_y()){
                        return h;
                }
                int m = (h + l)/2;
                if((key >= a[m]->max_y())&&(key < a[m + 1]->max_y())){
                        return m;
                }
                else if((key <= a[m]->max_y())&&(key > a[m -1]->max_y())) {
                        return m ;
                }
                else if(key >= a[m + 1]->max_y() ) {
                        l = m;
                }
                else {
                        h = m;
                }
        }
        return -1;                //key not found
 }

pointer* next_point(pointer* P, line* L){
        if(L->l[0] == P){
                return L->l[1];
        }
        else{
                return L->l[0];
        }
}

line* next_line(line* L, pointer* p){
        if(p->parent[0] == L){
                return p->parent[1];
        }
        else{
                return p->parent[0];
        }
}

double det( vector<vector<double>> mat)
{
        double d = 0;
```

```
        int n = mat.size();
        int c, subi, i, j, subj;
        vector<vector<double>> submat(mat.size() - 1, vector<double>(mat.size() -1));
        if (n == 2)
        {
                return( (mat[0][0] * mat[1][1]) - (mat[1][0] * mat[0][1]));
        }
        else
        {
                for(c = 0; c < n; c++)
                {
                        subi = 0;
                        for(i = 1; i < n; i++)
                        {
                                subj = 0;
                                for(j = 0; j < n; j++)
                                {
                                        if (j == c)
                                        {
                                                continue;
                                        }
                                        submat[subi][subj] = mat[i][j];
                                        subj++;
                                }
                                subi++;
                        }
                        d = d + (pow(-1 ,c) * mat[0][c] * det(submat));
                }
        }
        return d;
}

int partition_crossing(vector<vector<int>> &A, int start, int end){

        int i = start +1;
        vector<int> piv = A[start];
        for(int j =  start + 1; j <= end; j++){
                if(A[j][0] < piv[0]){
                        swap(A[i], A[j]);
                        //cout<<j<<endl;
                        i+= 1;
                }
        }
        swap(A[start] , A[i -1]);
        return i -1;
}

void quick_sort_crossing(vector<vector<int>> &A,int start, int end){
        if(start < end){
                int piv_pos = partition_crossing(A,start, end);
                quick_sort_crossing(A,start,piv_pos - 1);
                quick_sort_crossing(A,piv_pos + 1, end);
        }
}

double cal_angle(double x, double y){
        double a = x/ y;
        double theta;
        if( x >= 0 && a >= 0){
                theta = atan(a);
        }
        else if( x < 0 && a < 0){
```

```
                theta = PI + atan(a);
        }
        else if(x < 0 && a >= 0){
                theta = PI + atan(a);
        }
        else{
                theta = (2.0*PI) + atan(a);
        }
        return theta;
}

#endif // UTIL_H
```

main.cpp

```
#include <util.h>

#define RST  "\x1B[0m"
#define KRED  "\x1B[31m"
#define KGRN  "\x1B[32m"
#define KYEL  "\x1B[33m"
#define KBLU  "\x1B[34m"
#define KMAG  "\x1B[35m"
#define KCYN  "\x1B[36m"
#define KWHT  "\x1B[37m"

#define FRED(x) KRED x RST
#define FGRN(x) KGRN x RST
#define FYEL(x) KYEL x RST
#define FBLU(x) KBLU x RST
#define FMAG(x) KMAG x RST
#define FCYN(x) KCYN x RST
#define FWHT(x) KWHT x RST

#define BOLD(x) "\x1B[1m" x RST
#define UNDL(x) "\x1B[4m" x RST

int main(int argc, char**argv){
        vector<pointer*> Points(1000);
        vector<line*> lineset(1000);
        read(argv[1],Points, lineset);                                  //reading the
file
        //cout<<Points.size()<<" "<<lineset.size()<<endl;
        vector<line*> line2 = lineset;
        ofstream filej;
        filej.open("shape.fl");                                 //opening the file to
write the data
        quick_sort_lineS(line2,0,line2.size() - 1);
        filej<<line2.size()<<endl;
        /*for(int i = 0; i < Points.size(); i++){
                for(int j = 0; j < Points[i]->parent.size(); j++){
                        cout<<Points[i]->parent[j]<<" ";
                }
                cout<<endl;
        }*/
        for(int i = 0; i < line2.size();i++){
```

```
                //cout<<line2[i]->min_x()<<" "<<line2[i]->max_x()<<" "<<line2[i]-
>min_y()<<" "<<line2[i]->max_y()<<endl;
                filej <<line2[i]->l[0]->k[0]<<" "<<line2[i]->l[0]->k[1] <<" "<<line2[i]-
>l[1]->k[0] <<" "<<line2[i]->l[1]->k[1] <<endl;
        }
        /* we will link the lines which are connected to each other
         * through this we will make arc for the connections
         */
        /*Node* head = new Node();
        int linked_point = 0;
        while(linked_point < line2.size()){
                if(linked_point == 0){
                        line* current = line2[0];

                        line* pointN =
                }
        }
*/

        //filej<<endl<<endl;
        //cout<<endl<<"File arranged\n";
        for(int i= 0; i < line2.size(); i++){
                ofstream filei;
                filei <<line2[i]->l[0]->k[0]<<" "<<line2[i]->l[0]->k[1] <<" "<<line2[i]-
>l[1]->k[0] <<" "<<line2[i]->l[1]->k[1] <<endl;

                filei.open("line" + std::to_string(i) + ".fl");
                int me = line2.size();
                double ko = line2[i]->max_x();
                int j = binarySearch_min_x(line2,i,me - 1 , ko);
                //cout<<"first binary search "<<j<<" "<<i<<endl;
                if( j >= 0){
                        vector<line*>::const_iterator first = line2.begin() + i+1;
                        vector<line*>::const_iterator last = line2.begin() + j+1;
                        vector<line*> line3(first, last);
                        /*cout<<"list from fist search\n";
                        for(int in = 0; in < line3.size(); in++){
                                cout<<line3[in]->min_x()<<" "<<line3[in]->max_x()<<endl;
                        }*/
                        quick_sort_lineSminy(line3,0, line3.size() - 1);

                        for(int in = 0; in < line3.size() ; in++){
                                //cout<<i<<" "<<KRED<<line3[in]->min_x()<<"
"<<KGRN<<line3[in]->max_x()<<" "<<KBLU<<line3[in]->min_y()<<" "<<KMAG<<line3[in]-
>max_y()<<RST<<endl;
                        }
                        int fi = binarySearch_min_y(line3,0,line3.size() - 1, line2[i]-
>max_y());
                        //cout<<"for"<<fi<<endl;
                        if(fi >= 0){
                                //cout<<i<<" "<<j<<" "<<fi<<" "<<line3[fi]->min_y()<<"
"<<line3[fi]->max_y()<<" ";
                                /*if( fi < line3.size() - 1 ){
                                        cout<<line3[fi]->min_y();

                        quick_sort_lineSminy(line3,0, line3.size() - 1);
                        filei<<line3.size()<<endl;
        }*/
                                //cout<<"\nafter y bunary search\n";
                                for(int fl1 = 0; fl1 < line3.size(); fl1++){
```

```
                                        //cout<<KRED<<line3[fl1]->min_x()<<"
"<<KGRN<<line3[fl1]->max_x()<<" "<<KBLU<<line3[fl1]->min_y()<<" "<<KMAG<<line3[fl1]-
>max_y()<<RST<<endl;
                                        filei <<line3[fl1]->l[0]->k[0]<<" "<<line3[fl1]-
>l[0]->k[1]<<" "<<line3[fl1]->l[1]->k[0]<<" "<<line3[fl1]->l[1]->k[1]<<endl;
                                }
                                //cout<<endl<<"| ";
                                first = line3.begin();
                                last = line3.begin() + fi+1;
                                vector<line*> line4(first, last);
                                /*cout<<"-------------\n Printing from line4\n";
                                for(int in = 0; in < line4.size() ; in++){
                                        cout<<KRED<<line4[in]->min_x()<<"
"<<KGRN<<line4[in]->max_x()<<" "<<KBLU<<line4[in]->min_y()<<" "<<KMAG<<line4[in]-
>max_y()<<RST<<endl;
                                }*/
                                //cout<<line4.size()<<" ";
                                filei<<line4.size()<<endl;
                                quick_sort_lineSmaxy(line4, 0, line4.size() -1);
                                //for(int lik = 0; lik < line4.size(); lik++){
                                //      cout<<line4[lik]->max_y()<<" ";
                                ///}
                                //cout<<"|"<<line2[i]->min_y()<<"::| ";
                                int tfi = binarySearch_max_y(line4, 0, line4.size() - 1,
line2[i]->min_y());
                                //cout<<"tfi value "<<tfi<<endl;
                                if(tfi >= -1){
                                        //cout<<"from here "<<tfi<<" "<<line4.size() -
1<<endl;
                                        int k = line4.size() - 1;
                                        if(tfi == -1){
                                                tfi = 0;
                                        }

                                        if(tfi < k){
                                                //cout<<line4.size() - tfi<<endl;
                                                for(int fik  = tfi; fik < line4.size();
fik++){
                                                        //cout<<"file "<<line4[fik]-
>max_y()<<" "<<line4[fik]->min_y()<<endl;
                                                        filei <<line4[fik]->l[0]-
>k[0]<<" "<<line4[fik]->l[0]->k[1]<<" "<<line4[fik]->l[1]->k[0]<<" "<<line4[fik]->l[1]-
>k[1];
                                                        pointer* k1 =
lineIntersection(line2[i][0], line4[fik][0]);
                                                        if(k1 != NULL){
                                                                if((k1->k[0] - line2[i]-
>min_x()) > 0.000001 && (k1->k[0] - line2[i]->max_x())< 0.000001 && (k1->k[1] -
line2[i]->min_y()) > 0.000001 && (k1->k[1] - line2[i]->max_y()) < 0.000001){
                                                                        if((k1->k[0] -
line4[fik]->min_x()) > 0.000001 && (k1->k[0] - line4[fik]->max_x()) < 0.000001 && (k1-
>k[1] - line4[fik]->min_y()) > 0.000001 && (k1->k[1] - line4[fik]->max_y()) < 0.000001){

//cout<<"inte "<< k1->k[0]<<" "<<k1->k[1]<<" "<<line2[i]->min_x()<<endl;
                                                                                filej<<"
"<< k1->k[0]<<" "<<k1->k[1]<<" "<<endl;
                                                                                k1-
>parent.insert(k1->parent.end(), line2[i]);
                                                                                k1-
>parent.insert(k1->parent.end(), line4[fik]);
```

```
line2[i]->IntersectionPoints.insert(line2[i]->IntersectionPoints.end(), k1);

line4[fik]->IntersectionPoints.insert(line4[fik]->IntersectionPoints.end(), k1);
                                                                        }
                                                                        else{

filei<<endl;
                                                                        }

                                                                }
                                                                else{
                                                                        filei<<endl;
                                                                }
                                                        }
                                                        else{
                                                                filei<<"file"<<endl;
                                                        }
                                                }
                                        }
                                }
                        }
                        filei.close();
                }
                filei.close();
                //filej.close();
                //creating arc from

                //filek.close();
                cout<<endl<<endl;
        }
        cout<<"done\n"<<endl;
        cout<<"done\n"<<endl;
        ofstream filek;
        filek.open("arc.fl");
        cout<<"done\n";
        line* fl = line2[0];
        pointer* p = line2[0]->l[0];
        filek<<"p "<<p->k[0]<<" "<<p->k[1]<<endl;
        line* lp1 = fl;
        pointer* k1 = p;
        vector<pointer*> k2;
        cout<<"hello\n";
        vector<arc> arc_list(1);
        cout<<arc_list[0].size()<<endl;
        cout<<lp1<<endl;
        arc_list[0].insert(arc_list[0].end(), lp1);
        int n_arc = 0;
        int n_line = 0;
        cout<<"hik\n";
        for(int i = 0; i < 1000; i++){
                lp1->arc_p1 = k1;
                k1 = next_point( k1, lp1);
                lp1->arc_p2 = k1;
                filek<<"p "<<k1->k[0]<<" "<<k1->k[1]<<endl;
                if(k1 == p){
                        cout<<endl<<i<<endl;
                        break;
                }
                if(lp1->IntersectionPoints.size() != 0){
                        for(int j =0 ;j < lp1->IntersectionPoints.size(); j++){
```

```
                                filek<<"o "<<lp1->IntersectionPoints[j]->k[0]<<" "<<lp1-
>IntersectionPoints[j]->k[1]<<endl;
                                arc k_arc(1);
                                k_arc[0] = lp1;
                                arc_list.insert(arc_list.end(), k_arc);
                                n_arc++;
                        }
                }
                lp1 = next_line(lp1, k1);
                arc_list[n_arc].insert(arc_list[n_arc].end(), lp1);
        }
        int last = arc_list.size();
        int llast = arc_list[last-1].size();
        for(int i = 0; i < arc_list[last -1].size(); i++){
                arc_list[0].insert(arc_list[0].begin(), arc_list[last -1][llast - 1-i]);
        }
        arc_list.pop_back();
        filek.close();
        ofstream filel;
        filel.open("arc2.fl");

        for(int i = 0; i < arc_list.size(); i++){
                for(int j =0; j < arc_list[i].size(); j++){
                        filel<<i<<" "<<arc_list[i][j]->arc_p1->k[0]<<" "<<arc_list[i]
[j]->arc_p1->k[1]<<" "<<arc_list[i][j]->arc_p1->k[2]<<endl;
                }
        }
        filel.close();
        vector<arc_info> arc_list3(arc_list.size());
        for(int i = 0; i < arc_list.size(); i++){
        //      cout<<arc_list[i].size()<<endl;
        }
        cout<<"done2"<<endl;
        vector<intersect_info> i_info(arc_list.size());
        for(int i =0; i < arc_list.size(); i++){
                if(arc_list[i][0]->IntersectionPoints.size() > 0){
                        for( int j = 0; j< arc_list[i][0]->IntersectionPoints.size() ;
j++){
                                pointer* p = Z_val(*arc_list[i][0], *(arc_list[i][0]-
>IntersectionPoints[j]));
                                i_info[i].point = (arc_list[i][0]-
>IntersectionPoints[j]);
                                pointer* p1 = p;
                                if(arc_list[i][0]->IntersectionPoints[j]->parent[0] ==
arc_list[i][0]){
                                        p = Z_val(*(arc_list[i][0]-
>IntersectionPoints[j]->parent[1]), *(arc_list[i][0]->IntersectionPoints[j]));
                                }
                                else{
                                        p = Z_val(*(arc_list[i][0]-
>IntersectionPoints[j]->parent[0]), *(arc_list[i][0]->IntersectionPoints[j]));
                                }
                                pointer* p2 = p;
                                cout<<"b "<<arc_list[i].size()<<" "<<p1->k[2]<<" "<<p2-
>k[2]<<endl;
                                if(p1->k[2] > p2->k[2]){
                                        i_info[i].up = &(arc_list[i]);
                                        if(i == 0){

arc_list3[i].link(&arc_list3[arc_list.size() - 1]);
```

```
                                        //arc_list3[i].head =
arc_list3[arc_list.size() - 1].head;
                                        //arc_list3[i].headindx  =
arc_list3[arc_list.size() - 1].headindx;
                                        //arc_list2.insert(arc_list2.end(),
arc_list[arc_list.size()  - 1 ]);
                                }
                                else{
                                        arc_list3[i].link(&arc_list3[i - 1]);
                                        //arc_list3[i].head = arc_list3[i -
1].head;
                                        //arc_list3[i].headindx  = arc_list3[i -
1].headindx;
                                }
                                cout<<"a "<<arc_list[i].size()<<" "<<p1->k[2]<<"
"<<p2->k[2]<<endl;
                                //arc_list2.insert(arc_list2.end(),
arc_list[i]);
                                //i++;
                                arc_list3[i].indx  = -1;
                        }
                        else{
                                i_info[i].down = &(arc_list[i]);
                                arc_list3[i].head = &arc_list[i];
                                arc_list3[i].indx  = i;
                                arc_list3[i].headindx  = i;
                                arc_list3[i].link(NULL);
                        }
                }
        }
        filel.open("arc3.fl");
        vector<int> arm;
        int m = 0;
        for(int i = 0; i< i_info.size(); i++){
                cout<<i<<" "<<i_info[i].up<<" a "<<i_info[i].down<<" b
"<<i_info[i].point<<" "<<endl;
        }
        vector<vector<int>> point_up_down(0);
        for(int i = 0; i< i_info.size(); i++){
                for(int j = i + 1; j < i_info.size();j++){
                        if(i_info[i].point == i_info[j].point){
                                vector<int> l(2);
                                if(i_info[i].up == NULL){
                                        l[0] = i;
                                        l[1] = j;
                                        cout<<"down ";
                                }
                                else{
                                        l[0] = j;
                                        l[1] = i;
                                        cout<<"Up ";
                                }
                                point_up_down.insert(point_up_down.end(), l);
                                cout<<i<<" "<<j<<endl;;
                        }
                }
        }
```

```
        for(int i = 0; i < point_up_down.size(); i++){
                for(int j = 0; j < point_up_down[i].size(); j++){
                        cout<<point_up_down[i][j]<<" ";
                }
                cout<<endl;
        }
        vector<arc> arc_list2(0);
        vector<vector<int>> fil(arc_list3.size(),vector<int>(1,-1));
        cout<<fil.size()<<endl;
        for(int i = 0; i< arc_list.size(); i++){
                cout<<i<<" "<<arc_list3[i].indx<<" head Indx
"<<arc_list3[i].headindx<<endl;   //index head

                if( arc_list3[i].head == &arc_list[i])
                {
                        fil[i][0] = i;
                        arm.insert(arm.end(), i);
                        m++;
                }
                else{
                        cout<<"heiuh "<<i<<" dewd"<<arc_list3[i].head<<endl;
                        arc_list3[i].head->insert( arc_list3[i].head->end(),
arc_list[i].begin(), arc_list[i].end());
                        cout<<"heiuh"<<endl;

fil[arc_list3[i].headindx].insert(fil[arc_list3[i].headindx].end(), i);

                }
        }

        cout<<"uuihiwhie\n";
        for(int i = 0; i < arm.size(); i++){
                arc_list2.insert(arc_list2.end(), arc_list[arm[i]]);
        }
        cout<<"size arc "<<arc_list2.size()<<endl;

        vector<arc> arc_list4(0);
        for(int i = 0; i < arc_list2.size(); i++){
                if(arc_list2[i].size() > 0){
                        arc_list4.insert(arc_list4.end(), arc_list2[i]);
                }
                for(int j = 0; j < arc_list2[i].size(); j++){
                        //filel<<i<<" "<<arc_list2[i][j]->arc_p1->k[0]<<"
"<<arc_list2[i][j]->arc_p1->k[1]<<" "<<arc_list2[i][j]->arc_p1->k[2]<<endl;
                }
        }
        for(int i = 0; i < arc_list4.size(); i++){
                for(int j = 0; j < arc_list4[i].size(); j++){
                        filel<<i<<" "<<arc_list4[i][j]->arc_p1->k[0]<<" "<<arc_list4[i]
[j]->arc_p1->k[1]<<" "<<arc_list4[i][j]->arc_p1->k[2]<<endl;
                }
        }

        int fis = 0;
        vector<vector<int>> fisv(0);
        for(int i = 0; i <  fil.size(); i++){

                if(fil[i][0] >= 0){
                        fisv.insert(fisv.end(),vector<int>(0));
                        cout<<fis<<" :- ";
                        for(int j = 0; j < fil[i].size(); j++){
                                fisv[fis].insert(fisv[fis].end(),fil[i][j]);
```

```
                        cout<<fil[i][j]<<" ";
                }
                fis++;
                cout<<endl;
        }
}

cout<<"fisv: \n";
for(int i = 0; i < fisv.size(); i++){
        cout<<i<<" : ";
        for(int j = 0; j < fisv[i].size(); j++){
                cout<<fisv[i][j]<<" ";
        }
        cout<<endl;
}

filel.close();

vector<vector<int>> nePoint = point_up_down;
for(int i = 0; i < point_up_down.size(); i++){
        for(int j =0; j < fisv.size(); j++){
                if(fisv[j][0] == point_up_down[i][0]){
                        nePoint[i][0] = j;
                        break;
                }
        }

        for(int j =0; j < fisv.size(); j++){
                int oi = 0;
                for(int k = 1; k < fisv[j].size(); k++){
                        if(fisv[j][k] == point_up_down[i][1]){
                                nePoint[i][1] = j;
                                oi = 1;
                                break;
                        }
                }
                if(oi == 1){
                        break;
                }
        }
}
for(int i = 0; i < nePoint.size(); i++){
        for(int j = 0;j < nePoint[i].size(); j++){
                cout<<nePoint[i][j]<<" ";
        }
        cout<<endl;
}
quick_sort_crossing(nePoint,0,nePoint.size() -1);

//sorted order of the point of intersection;
cout<<"\n\n";

for(int i = 0; i < nePoint.size(); i++){
        for(int j = 0;j < nePoint[i].size(); j++){
                cout<<nePoint[i][j]<<" ";
        }
        cout<<endl;
}

cout<<endl;

vector<vector<double>> matrix(arc_list4.size(),
vector<double>(arc_list4.size()));
```

```
        double t = 5;
        filel.open("mati.fl");

        for(int i = 0; i < matrix.size(); i++){
                double y = (arc_list4[i][0]->arc_p2->k[1] - arc_list4[i][0]->arc_p1-
>k[1]);
                double x = (arc_list4[i][0]->arc_p2->k[0] - arc_list4[i][0]->arc_p1-
>k[0]);
                double theta = atan2(y,x);
                if(arc_list4[i][0]->IntersectionPoints[0]->parent[0] == arc_list4[i][0])
{
                        y = arc_list4[i][0]->IntersectionPoints[0]->parent[1]->arc_p2-
>k[1] - arc_list4[i][0]->IntersectionPoints[0]->parent[1]->arc_p1->k[1];
                        x = arc_list4[i][0]->IntersectionPoints[0]->parent[1]->arc_p2-
>k[0] - arc_list4[i][0]->IntersectionPoints[0]->parent[1]->arc_p1->k[0];
                }
                else{
                        y = arc_list4[i][0]->IntersectionPoints[0]->parent[0]->arc_p2-
>k[1] - arc_list4[i][0]->IntersectionPoints[0]->parent[0]->arc_p1->k[1];
                        x = arc_list4[i][0]->IntersectionPoints[0]->parent[0]->arc_p2-
>k[0] - arc_list4[i][0]->IntersectionPoints[0]->parent[0]->arc_p1->k[0];
                }
                double theta2 = atan2(y,x);
                double k = theta2 - theta;
                if(k > PI){
                        k = -(k -PI);
                }
                if(k < -PI){
                        k = -(k + PI);
                }
                cout<<theta<<" "<<theta2<<endl;
                filel<<arc_list4[i][0]->arc_p2->k[0]<<" "<<arc_list4[i][0]->arc_p2-
>k[1]<<" "<<arc_list4[i][0]->arc_p2->k[2]<<" "<<arc_list4[i][0]->arc_p1->k[0]<<"
"<<arc_list4[i][0]->arc_p1->k[1]<<" "<<arc_list4[i][0]->arc_p1->k[2]<<endl;
                if(arc_list4[i][0]->IntersectionPoints[0]->parent[0] == arc_list4[i][0])
{
                        filel<<arc_list4[i][0]->IntersectionPoints[0]->parent[1]-
>arc_p2->k[0]<<" "<<arc_list4[i][0]->IntersectionPoints[0]->parent[1]->arc_p2->k[1]<<"
"<<arc_list4[i][0]->IntersectionPoints[0]->parent[1]->arc_p2->k[2]<<" "<<arc_list4[i]
[0]->IntersectionPoints[0]->parent[1]->arc_p1->k[0]<<" "<<arc_list4[i][0]-
>IntersectionPoints[0]->parent[1]->arc_p1->k[1]<<" "<<arc_list4[i][0]-
>IntersectionPoints[0]->parent[1]->arc_p1->k[2]<<endl;
                }
                else{
                        filel<<arc_list4[i][0]->IntersectionPoints[0]->parent[0]-
>arc_p2->k[0]<<" "<<arc_list4[i][0]->IntersectionPoints[0]->parent[0]->arc_p2->k[1]<<"
"<<arc_list4[i][0]->IntersectionPoints[0]->parent[0]->arc_p2->k[2]<<" "<<arc_list4[i]
[0]->IntersectionPoints[0]->parent[0]->arc_p1->k[0]<<" "<<arc_list4[i][0]-
>IntersectionPoints[0]->parent[0]->arc_p1->k[1]<<" "<<arc_list4[i][0]-
>IntersectionPoints[0]->parent[0]->arc_p1->k[2]<<endl;
                }
                filel<<endl;
                if(nePoint[i][1] == i){
                        matrix[i][i] = -1;
                }
                if(i + 1 <= matrix.size() - 1){
                        if(nePoint[i][1] == i+1){
                                matrix[i][i+1] = -1;
                        }
                }
                else{
```

```
                        if(nePoint[i][1] == 0){
                                matrix[i][0] = -1;
                        }
                }
                if(k > 0){
                        if((i-1) >= 0){
                                matrix[i][(i - 1)] = 1;
                        }
                        matrix[i][i] = -1*t;
                        matrix[i][nePoint[i][1]]  = t -1;
                        cout<<i<<" a "<<nePoint[i][1]<<" "<<(i - 1)%matrix.size()<<endl;
                }
                else{
                        if((i-1) >= 0){
                                matrix[i][(i - 1)] = -1*t;
                        }
                        matrix[i][i] = 1;
                        matrix[i][nePoint[i][1]]  = t -1;
                        cout<<i<<" b "<<nePoint[i][1]<<" "<<(i - 1)%matrix.size()<<endl;
                }
        }
        filel.close();

        for(int i = 0; i < matrix.size(); i++){
                for(int j = 0; j < matrix[i].size();j++){
                        cout<<matrix[i][j]<<" ";
                }
                cout<<endl;
        }
        return 0;
}
```

a4.py

```
#!/usr/bin/env python

import sys
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib.patches import FancyArrowPatch
from mpl_toolkits.mplot3d import proj3d
from matplotlib.widgets import Slider, Button, RadioButtons
import random as r

class Arrow3D(FancyArrowPatch):
    def __init__(self, xs, ys, zs, *args, **kwargs):
        FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwargs)
        self._verts3d = xs, ys, zs

    def draw(self, renderer):
        xs3d, ys3d, zs3d = self._verts3d
        xs, ys, zs = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
        self.set_positions((xs[0],ys[0]),(xs[1],ys[1]))
        FancyArrowPatch.draw(self, renderer)
        #print(renderer)

def Rx(phi):
    return np.array([[1, 0, 0],
                     [0, np.cos(phi), -np.sin(phi)],
```

```
                    [0, np.sin(phi), np.cos(phi)]])

def Ry(theta):
    return np.array([[np.cos(theta), 0, np.sin(theta)],
                    [0, 1, 0],
                    [-np.sin(theta), 0, np.cos(theta)]])

def Rz(psi):
    return np.array([[np.cos(psi), -np.sin(psi), 0],
                    [np.sin(psi), np.cos(psi), 0],
                    [0, 0, 1]])

with open(sys.argv[1]) as f:
        lines = f.readlines()
mpl.rcParams['legend.fontsize'] = 10
fig = plt.figure()
ax = fig.gca(projection='3d')
a = 0
k1 = [[]]
k2 = [[]]
k3 = [[]]
b = 0
for i in lines:
        k =  i.split();
        if(int(k[0]) == a):
                k1[b].append(float(k[1]))
                k2[b].append(float(k[2]))
                k3[b].append(float(k[3]))
        else:
                k1.append([])
                k2.append([])
                k3.append([])
                a = a+1
                b = b+1
                k1[b].append(float(k[1]))
                k2[b].append(float(k[2]))
                k3[b].append(float(k[3]))

a0 = 5
f0 = 3
axcolor = 'lightgoldenrodyellow'
axi =[]
#for i in range(b+1):
#       axi.append( plt.axes([0.25, 0.02*i + 0.1, 0.65, 0.01],))
print(len(k1[0]))

si = []
#for i in range(b+1):
#       si.append(Slider(axi[i], 'k'+str(i), 0.1, len(k1[i]) - 2, valinit=f0))
a = []
for i in range(len(k1)):
        f = '%06x' % r.randrange(0xffffff)
        #plt.plot(k1[i],k2[i], color="#"+f, label="feo "+str(i))
        a.append(Arrow3D([k1[i][4],k1[i][5]], [k2[i][4], k2[i][5]], [k3[i][4],k3[i][5]]
, mutation_scale=20, arrowstyle='->', color="#"+f))
        ax.add_artist(a[i])
        k, = ax.plot(k1[i],k2[i],k3[i], color="#"+f, label="segment "+str(i))
        print(k)

ax.view_init(elev = 90, azim= 0)
plt.savefig(sys.argv[2]+"e90.png")
ax.view_init(elev = 45)
```

```
plt.savefig(sys.argv[2]+"e45.png")
#def update(val):
#       ai = []
#       for i in range(len(a)):
#               ai.append(int(si[i].val))
#               a[i]._verts3d = [k1[i][ai[i]], k1[i][ai[i] + 1]], [k2[i][ai[i]],k2[i]
[ai[i] + 1]], [k3[i][ai[i]],k3[i][ai[i] + 1]]

#for i in range(len(si)):
#       si[i].on_changed(update)

print k
plt.legend()
plt.show()
```