Searching for Handedness in Large Data

Summer project

Mark Jenei, UCL

[4]

Supervisors: Prof. Ian Robinson, Chritopher Lynch

Summer of 2015

Abstract

Many biological systems have a well-defined chirality (handedness), which affects their biological behaviour. With novel super-resolution methods, it is now possible to image *in vivo* chromosomes with high resolution that allows deeper analysis of the structure of a chromatin. Nucleosomes within the core of cells may or may not build up a 30 nm fibre within the chromatin. The aim of the experiment is to figure out whether this 30 nm exists and if it has a handedness. For this purpose a computational method was used described in this report to analyse data collected with direct stochastical optical reconstruction microscopy (dSTORM).

Introduction

Within the cells biological information is stored in RNA and DNA molecules. These wrap around protein molecules (histones) to form the chromatin within the cell's nucleus. The first level of the major structures in the chromatin is the 10 nm "beads-on a string" fibres, where DNA molecules curl around a histone (Figure 1 [1]). The DNA spiral around the protein molecule is left-handed as opposed to the DNA molecule's right-handed helix.

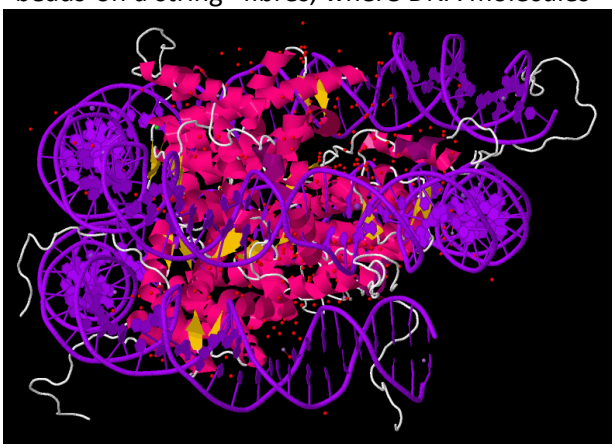

Figure 1

The 10 nm structure in the chromatin. DNA warps a histone with a left-handed curl.

There are two main phases in the cell cycle: inter – and metaphase. During interphase the chromatin is loosely packed within the nucleus allowing access to the DNA for reproduction purposes. On the other hand, during metaphase the chromosomes are most condensed and highly structured to protect the DNA molecules during mitosis (division of the nucleus).

Between the high level structure of the chromosomes and the 10 nm structure there was found a 30 nm structure as well, however it has only been seen *in vitro* circumstances and in intact chicken erytochrytes, but has not been observed *in vivo* human cells. With the new high-resolution imaging methods (briefly described below), it may be possible to get more information about *in vivo* human cells and the 30 nm structure.

Method

To record the position of atoms in the DNA the dSTORM super-resolution method was used [2]. The atoms were labelled using organic dies (Alesca fluor 647) and a reducing agent (Dithithreitol) was used to activate the fluorophores. Due to the stochastic manner of activation, labels within each other's vicinity are activated separately in time, allowing to resolve them and image separately. After the images were recorded a Gaussian has been fitted to each individual point-like light source. By finding the centre of the Gaussian, one can locate the atoms with high (order of nm) precision, giving the lateral (x, y) coordinates.

To determine the z coordinate each, image is split into two lobes, whose distance and relative orientation depends on the axial coordinate. Using a phase mask, a double helix pattern can be generated that rotates around the z axis as Figure 2 [3] shows. Comparing the resulted image with the calibration (done by a piezo-electric device) the z coordinate can be determined with precision in the same order as in the case of lateral coordinates. The midpoint between the two helices also describe the lateral position. The uncertainty in all three directions are typically in the order of 10-20 nm.

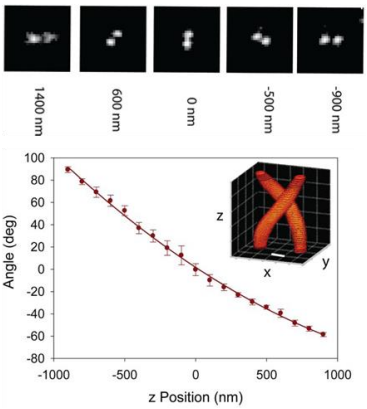

Figure 2
The determination of z coordinate

Theory

For handedness analyses, point quartets are constructed within the sample and connected with imaginary bonds. The range of length of these bond and the maximum angle between them are specified. Let's look at one quartet and label its points by A, B, C and D (see Figure 3).

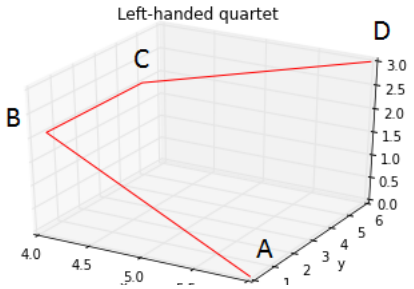

Figure 3
A point quartet selected from the sample with the imaginary bonds.

The main approach to handedness is that if the dihedral angle between the planes ABC and BCD within the quartet 0 and π the structure is said to be left-handed and right-handed for angles between π and 2π. The dihedral angles is measured the following way: imagine the four points in one plane, the dihedral angle grows from zero by rotating the CD bond anti-clockwise with respect to the BC bond.

The program described below selects many (or all, depending on the computational method used, see below) of these quartets found in the raw data and creates a dihedral angle distribution. The idea is, that if there's a chiral structure in the sample, the angle distribution will be biased (more angles in one region e.g. for left-handed structure in the 0- π region), hence one can tell if there's a chiral structure in the sample by looking at the dihedral angle distribution.

The Code

The program is based on 'while' loops. In python for a while loop there is a condition followed by an action as long as the condition is satisfied. For example if the condition is: while x < 5, the action will repeated as long x is less than five. Within the action one can change the parameters, for example the value of x, so at one point the condition is not met and the program stops performing the action written within the loop. A simple example is shown on Figure 4.

```
In [1]: x = 0

        while x < 3:
            print 'The current value of x is:', x
            x = x + 1

The current value of x is: 0
The current value of x is: 1
The current value of x is: 2
```

Figure 4
A simple loop

Below, the working of each cell is described in Handedness 3.0. Handedness 2.2(Quick) work in a very similar way, the few things that are different are outlined below.

Cell 1

The program analyses a dataset for a selected number of bond lengths. The first range of these is given by *r1* and *r2*, the lower and upper limit of bond lengths respectively. The *step* determines by how much these limits are raised, for example if *r1 = 3* and *r2 = 4* and *step* is set to be 2, the following limits of bond lengths once the analysis for this range is done will be 5 and 6, then 7 and 8 etc. *no* give the number of bond length ranges required in the full analysis, if for example *no* is set to be 5, the program will return 5 ranges of bond lengths. *Phi* is the maximum allowed angle between successive bonds (advised to set less then $\pi/2$, so the structure doesn't curl back into itself). *region* is the number of regions required to analyse. Usually it is set to a high number, since we are interested in all regions for a given bond length and not just the first few.

Cell 2

In this cell some modules are imported (built-in functions) and the data is loaded from a .txt file that has the following format: the x, y and z coordinates of one atom is in one row, separated by a space. **The data with which the program works has to be given in this cell:** in the 11th line the *tRNA.txt* file has to be replaced by *filename.txt*. The coordinates of the next atom is in the second row. The created *x, y* and *z* arrays contain the coordinates of the atoms in order, i.e. x = [x_0, x_1, … , x_N], y = [y_0, y_1, … , y_N], z = [z_0, z_1, … , z_N]. Once the coordinates are loaded, an array is created containing the coordinates of the atoms in a more desirable form as: V = [[x_0, y_0, z_0], [x_1, y_1, z_1], … , [x_N, y_N, z_N]]. The number of points and the range of coordinates is written below the cell.

Cell 3

In this cell some minor functions are defined. In python it can be done using the *def* function followed by the name of the function and the variables within it. At the end it returns whatever is specified within *def*. A simple example is shown on Figure 5. The argument of the function contains local variables, they're valid only within the *def* function (e.g. x and y can be used later on independent from the function *sum*).

```
def sum(x,y):
    '''Adds two number'''
    return x + y

print sum(3,7)
```

```
10
```

Figure 5
A simple function defined in python

Cell 4

For constructing the point quartets used for the dihedral angle calculations, point pairs (i.e. the BC bonds) are created first, matching the bond length criterion. For this, a function is defined with local variables *r1* and *r2*. At first, the start time is recorded (*startBC*), the variables used to control the while loop are set to zero, and 'containers' are introduces, in which the appropriate point pairs are stored. In the loop, P is the array of points within the observed region (see below, Cell 8). At first a point (B) is selected from this region by setting B = P[i]. Since i is set to zero (loop variable), B will be the first point (python start counting from zero instead of one) of the array P. After B is found another loop is introduced that runs over all the points of P with higher indices than B (so if B was the fifth point in the array P, it'll check all the points starting from the sixth) and calculates the distance between B and this other point. If the bond length criteria is matched, the point is appended to the container C, which, after the loop with loop variable *j* has run over the points of P, contains all the C point matching to the one selected B point. The constructed point pairs are then stored in the BC container as the following: [[[x_B,y_B,z_B], [x_{C1},y_{C1},z_{C1}]], [[x_B,y_B,z_B], [x_{C2},y_{C2},z_{C2}]], …], a three level array. Once all the corresponding C points matching to this particular point B have been found, container C is emptied and the loop moves to the next point B, until all the points within P have been investigated.

The function returns the array BC, which contains all the appropriate point pairs within the sample, the time taken to construct these pairs is printed, and the length of the BC array corresponding to this range of bond lengths is stored in the container *statBC*.

Cell 5

In this cell a function is defined, which finds appropriate points A and D for every BC point pairs. Two loop variables are introduced: l and m, the first for selecting the point pairs from the BC array, the second to select a point (*a*) from P. Once they are selected the vectors representing the bonds (*va, vb* and *vc* respectively) are defined by subtracting the coordinates of the points from each other. For example the vector *va* (pointing from A to B) will be constructed as *BC[l][0] – a*, where *BC[l][0]* is the (l-1)th point pair's first (0th) element, i.e. a point B and *a* equals *P[m]*, i.e. the (m-1)th point in the region *P*. For a point *a* to become an appropriate point in the point quartet ABCD there are two criterion to match: first it has to be within the range of distances *r1, r2* measured from B/C and second, the angle between the created bonds (*vb* and *va*/*vc*) has to be less then *phi* defined in the first cell. If both of these criterion are matched point *a* is appended to the container A or D depending which pair of constrains are satisfied (A or D criterion).
After all the proper A and D points have been found for a particular BC point pair, all the quartets are appended to the ABCD container that the function returns. The number of quartets for this bond length is stored in the *statABCD* container.
Once all the quartets (so for one BC pair all proper A and D points) have been found, the A and D containers are emptied (*A = [ ], D = [ ]*) for the next BC point pair.

Cell 6

The *angles* function calculates the dihedral angles for all of the ABCD quartets. The clock is started and a loop variable (*q*) is introduced to select the quartets from the array ABCD. The container *dtheta* stores the dihedral angles for all points ABCD[q] within the current region P. Once the quartet is selected (ABCD[q]) the dihedral angle is calculated as the following:

- The bonds are created as vector pointing from A to B (*va2*), B to C (*vb2*) and C to D (*vc2*) via subtracting the point's coordinates from each other.

- The normalized cross products of the neighbouring bonds are created, and the dot product of the two cross product is taken: `dot(norm(cross(va2,vb2)),norm(cross(vb2,vc2)))`.

- The dihedral angle is calculated by taking the inverse cosine of the dot product above. However, one inverse cosine value corresponds to two angles: α and 2π-α.

- To resolve the two angles *sign* is introduced as the following:

$$sign = round\left(\frac{vb2\ .\ ((va2\ x\ vb2)\ x\ (vb2\ x\ vc2))}{|vb2\ .\ ((va2\ x\ vb2)\ x\ (vb2\ x\ vc2))|}\right)$$

  sign can take two values, 1 and -1. If *sign* is 1 then the dihedral angles is in the region [π;2π] so *theta2* (the particular dihedral angle for this quartet) will be 2π – arccos(*expression above*), but if *sign* is -1 theta 2 is simply arccos(*expression above*). The selection is acquired by f: f[*sign*] selects either the -1st or the 1st element of the array f. In python for this array the -1st element is 0 (the third element) and the 1st element is 1 (the 2nd element, python starts counting at 0). Depending on the selected element of f in the expression for *theta2* in the first term the 2π is kept or omitted just as the minus sign (-1x) at the beginning of the second term.

Once the dihedral angle is calculated it is appended to the container *theta* that stores all dihedral angles in the space of the sample and to the container *dtheta* that stores angles only in the current region of points P. Depending in which region *theta2* falls, it is also appended to the *RHS, LHS* containers, which are used for the chirality index calculation at the end of the analysis of this bond length. Once all angles in this region are calculated the histograms showing the angle distributions (with different bin numbers) are plotted and *q* is raised by one to select the next quartet ABCD[*q+1*].

Cell 7

This cell simply visualizes the data points in the entire sample. For 3D projection the `ax = fig.add_subplot(111, projection='3d')` is used.

Cell 8

This is the cell where the computation actually happens. First a loop variable for the different bond lengths (*counter*) is introduced, then containers are set up for the chirality index plotting (*diffnorm* and *R*) and for the statistical analysis (that contains the important information about each region). The *stat* containers are to be handled in the following way: container *stat* contains every information about one region, so *stat[0]* will have the following elements of the 0^{th} region in order: *[r1,r2,x0,x0+ax,y0,y0+ay,z0,z0+az,len(P),len(BC),len(ABCD),dt],* i.e. for one region the bond length range (*r1*, *r2*) the coordinates of the vertices of the cuboid region (following 6 element), how many points are in the region (*len(P)*), number of point pairs and quartets and the time taken to analyse this region (*dt*). The other *stat* containers contain one specific information for all the region, so for example *statBC[5]* will be the number of point pairs in the sixth region.
The sides of the cuboid regions in which the whole space is divided (starting from the origin) **can be set here.** The smaller the sides are the faster the program finishes but the less data (dihedral angles) we'll have.
A loop variable is introduced to select the regions (*reg*) and a three-level loop system is in charge for going through every region within the whole space. The loops are working as the following:

- A container is introduced that stores the points within this region (P)
- Another loop (with loop variable s) goes through all the points in the whole space of the dataset and examines if their coordinates satisfies the constraints need to belong to the region P, e.g. the x, y and z coordinates of the sixth point in the whole space is *x[5]*, *y[5]* and *z[5]*. If *x0 < x[5] < x0 + ax*, so the x coordinate of this point is between the endpoints of the side parallel with the x axis and if the same is true for the y and z coordinates, the point is appended to P. Once all the points are found in this region (we have P), the number of them is appended to *statP*.
- Now the Main Functions defined above are applied on the region P (*fBC*, *fABCD* and *angles*) to acquire the dihedral angle distribution. The particular pieces of information about the analysis are appended to the appropriate stat containers and the endpoints of the side of the container parallel with the x axis are displaced by *ax* (so *x0 = x0 + ax*), basically this line shifts the region by *ax* in the x direction
- If one of the endpoint exceeds the range of x coordinates in the sample, *x0* is reset to be at the predefined origin (the cuboid is back at the origin) and *y0* is set to *y0 + ay* that shifts the cuboid region by *ay* in the y direction. (followed by multiple x direction shifts again)
- Shifting in the z direction takes place at the same circumstances (when the endpoint of the side parallel to the y axis exceeds the sample's range of y coordinates).

Once the analysis of the whole region is finished the histograms representing the whole space of data points are plotted.

Cell 9

The graphs for the chirality index and the time taken for each bond length are plotted.

Differences in Handedness 2.2(Quick)

The difference in this version is that the program doesn't dived the space into regions where the analysis takes place, but takes the whole space. However, to make the computation faster, for a given BC point pair, it doesn't observe all point if it is applicable as an A or D point, but selects a number of random point and checks if they satisfy the criterion for being a point A or D.

The number of point selected for each point pair can be controlled in the first cell by *ran*. If there are *N* points in the sample, for every point pair the program will select *N/ran* points to investigate. Since the time taken to select the points A and D are almost the same as the whole analysis (this is by far the slowest part), if the number of points checked is reduced by a factor of *ran* so will be the length of the analysis.

The other (minor) difference is, since regions are not used, for one range of bond lengths there's only one set if histograms, hence the plotting of them is embodied in the *angles* function (in this program it is called *hABCD* function)

Length of computations

The time it takes to finish an analysis has been investigated for two variables. First the range of bond lengths have been varied. In theory (assuming a fairly constant density of atoms) the number of point pairs should grow with approximately the second power of the average bond length (the mean of the upper and lower limit of the bond length). Instead, the number of point pairs, hence the time taken for the analysis grows approximately linearly with the bond length. The number of point pairs and the time taken is strongly correlated, since for every point pair the program checks every point if it's suitable as a point A or D, and it takes most of the analysis concerning time. Figure 6 shows the time relation to the average bond length.

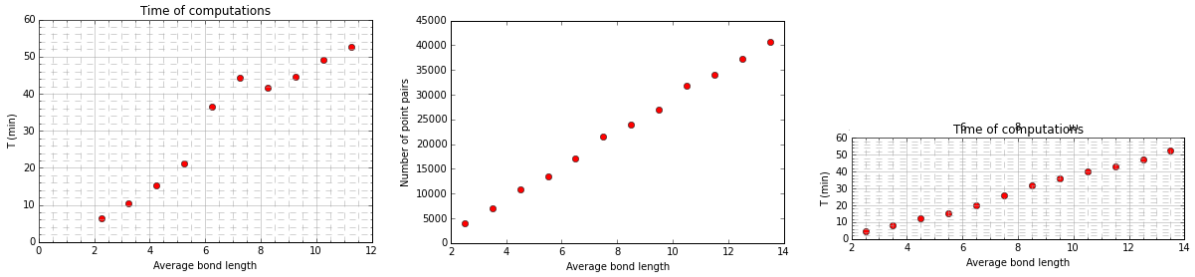

Figure 6

Time taken for a full analysis as a function of time. For the first graph a random point distribution in a cuboid region was used. At bond lengths 6.25 and 7.25 another application was run on the computer, hence the slower computation. At the second and the third graph a tRNA molecule was analysed. The number of point pairs and the time taken for each computation is strongly correlated. The time grows approximately linearly with bond length.

Secondly, the radius was kept constant and the size of the regions were varied. In this case a random point distribution was used in a cuboid region. As expected, the number of point pairs grows with the 3rd power of the side length (since the number of points grows like that as well), just as the time taken for the full analysis again as expected for the following reason: for a given region the time taken would grow with the sixth power of the side length (3rd power because of the number of point

pairs and then for every point pair it makes the analysis with all the points in the sample that also grows with the third power). However, in a given sample it takes less region to analyse if the side length is long and since the number of regions is in proportion with the reciprocal of the third power of the side length (volume), the overall time will be proportional with the third power of the side length. Figure 7 sums up the above written.

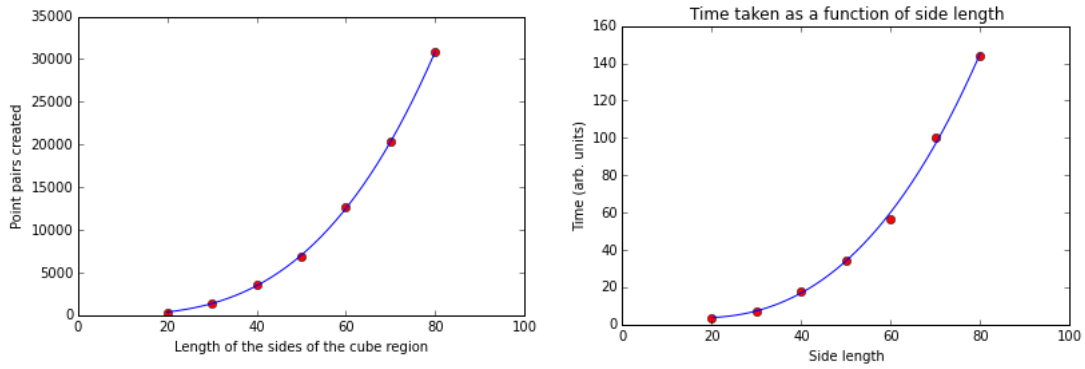

Figure 7

Point pairs created and time taken for full analysis as a function of the regions' side length with a fitted third order polynomial. In the second graph the computation was performed on one region and then scaled with the reciprocal of the third power of the side length (longer sides, less region required for the full analysis of the whole space).