Directionality Analysis

Kota Miura
Centre for Molecular and Cellular Imaging
EMBL Heidelberg
69120 Germany

miura@embl.de
http://cmci.embl.de

May 14, 2012

1 Protocol 1. Tracking by the ParticleTracker Plugin of ImageJ/Fiji

We use open source software Fiji for particle tracking. Fiji is a distribution of ImageJ, bundled with many useful plugins.

Get Image Files

Download and save the image files in your local computer. A file named **eb18_b.tif** is a sequence taken from single cultured cell labeled with eb1. We use this as the first sample to measure the directionality. Other two image sequences, **early.tif** and **late.tif** are time-lapse of *Drosophila* embryo from late cellularizatino to gastrulation phase. During this period cells become polarized. Each file was extracted from a longer single sequence. We study if there is bias in microtubule orientation, measured by movement direction of EB1 protein by tracking. We then try to compare the changes in the degree of bias in the movement direction using descriptive circular statistics.

Open the file in Fiji by

`[File > Open...]`

Examine the sequence using stack tools.

Explore stack functions.
Start animation, Stop animation, change frame rates, Manipulations...

Download ParticleTracker Plugin and Install

The plugin could be down loaded from

1. `http://www.mosaic.ethz.ch/Downloads/ParticleTracker`

To install the downloaded plugin, `[Plugins > Install...]`. You could also directly copy the file to plugins folder under Fiji directory and do `[Help > Refresh Menus]`

Set correct dimensions of the image by
`[Image > Properties...]`

...Image stacks are by default taken as a z-series and not t-series. Set Slices to 1, and Frames to appropriate size (number of frames).

Start the ParticleTracker plugin

Start the particleTracker by `[Plugins > Mosaic > ParticleTracker 2D/3D]`.

Study Dot Detection Parameter

This tracking tool has two parts. First, all dots in each frame are detected, and then dots in successive frames are linked. The first task then is to determine three parameters for dot detection. There are three parameters.

1. **Radius**
   Expected diameter of dot to be detected in pixels.
2. **CutOff**
   Cutoff level for the none-particle discrimination criteria, a value for each dot that is based on intensity moment order 0 and 2.
3. **Percentile**
   Larger the value, more particles with dark intensity will be detected. Corresponds to the area proportion below intensity histogram in the upper part of the histogram.

Try setting different numbers for these parameters and click ”Preview Detected”. Red circles appear in the image stack. You could change the frames using the slider below the button.

After some trials, set parameters to what you think is optimum.

Set Linking parameters

Two parameters for linking detected dots should be set.

1. **Link Range**
   . . . could be more than 1, if you want to link dots that disappears and reappears. If not, set the value to 1.
2. **Displacement**
   . . . expected maximum distance that dots could move from one frame to the next. Unit is in pixels.

After parameters are set, click ”OK”. Tracking starts.

Inspect the Tracking Results

When tracking is done, a new window titled ”Results” appears. At the bottom of the window, there are many buttons. Click ”Visualize all trajectories”, and then a duplicate of the image stack overlaid with trajectories will appear.

Select a region within the stack using rectangular ROI tool and then click ”Focus on Area”. This will create another image stack, with only that region. Since this image is zoomed, you could carefully check if the tracking was successful or not.

If you think the tracking was not successful, then you should reset all the parameters and do the tracking again.

Export the tracking results

To analyze the results in R, data should be saved as a file. To do so, first click ”All Trajectories to Table”. Results table will then created. In this results table window, select `[File > Save As...]` and save the file on your desktop. By default, file type extension is ”.xls”, excel format, but change this to ”.csv”. CSV stands for ”comma separated file”, and this is more classic but general data format which you could easily import in many software including R.

2 Protocol 2. Directionality Analysis Using R

R is an open source statistical analysis tool widely used in scientific community. Many packages are available as additional module. There are several interfaces available for R, and we use Rstudio in this practical course.

Exercise: A Very Short Introduction to R

To set a variable ”a” to 3,

```
a <- 3
```

Similarly, ”b” to 5,

```
b <- 5
```

To see the content of variable, simply type

```
a
```

Then you will see a print out in the console

```
[1] 3
```

To calculate,

```
a + b
```

then the print out will be

```
[1] 8
```

to store the result of calculation in ”c”

```
c <- a - b
```

Check the results

```
c
```

prints out

```
[1] -2
```

Down till here, we made only one value associated with a variable, but a variable could contain multiple numbers (we call it a ”vector”). First try the following

```
1:5
```

this will print out

```
[1] 1 2 3 4 5
```

similarly,

```
2:10
```

then the output

```
[1] 2  3  4  5  6  7  8  9 10
```

such numerical sequence could be stored in a variable

```
d <- 1:10
```

check the content of ”d”, a vector.

```
d
```

output will be

```
[1]  1  2  3  4  5  6  7  8  9 10
```

We now that the length of the vector ”d” is 10, but for the work you do in the following, you better know the command to know the length of vector.

```
length(d)
```

”length()” is a command that inspects the vector and returns its length, so the output will be

```
[1] 10.
```

Similarly,

```
length(a)
```

prints out

```
[1] 1
```

We could generate a vector in a bit more complex way by using command ”seq(start number, end number, increment)”.

```
da <- seq(1, 10, 0.7)
da
```

will print out

```
[1] 1.0 1.7 2.4 3.1 3.8 4.5 5.2 5.9 6.6 7.3 8.0 8.7
[13] 9.4
```

Installing Packages

The base package of R already has many useful functions, but when you have complex and specific task, you could either write your own script for that task, or you could also search for a package that does the job. In many cases, searching an appropriate package is more efficient than writing your own. We use following packages:

1. fisheyeR
2. CircStats
3. circular
4. plotrix

In Rstudio, you could see available packages in the ”Packages” tab in the right bottom panel. All the packages listed there could be simply loaded from your local R distribution. Many of packages which are not listed there should be downloaded from the internet and installed. Download and installing actually is not difficult, since there is a menu command for that purpose.

`[Tools > Install Packages]`
Type in the package name that you want to install in the second field, and simply click ”Install” will do all the job for you.

After installation if finished, check the ”packages” tab again and click the check box for the installed package. If the box is checked, you could use functions offered by the package.

In the following, command line interface in the left-bottom panel will be used. Command input field starts with a prompt ‘>’, but will be omitted in this textbook.

Loading tracking data

Load data by the following command.

```
ptdata <- read.csv("Z:/11EMBOcourse/trials/PTresults.csv")
```

The argument within the parenthesis is just an example path, and it should be adjusted according to where you have saved the file.

If you happen to have exported the data without changing the file extension, then the file ending should be ”.xls” and in that case, you have your data not in comma separated, but in tab-delimited file. To import such data, you need an option added.

```
ptdata <- read.csv("Z:/11EMBOcourse/trials/PTresults.xls", sep="\t")
```

The second argument explicitly states that the values are separated by tab (`\tab`).

If the import is successful, you will find data `ptdata` being listed in the "Workspace" tab in the right top column. Single clicking that data name in the list will open a table in the left-top panel showing the content of that data variable.

More traditional way of checking data content is from command line.

```
head(ptdata)
```

function `head()` command will print the first row in the data.

IMPORTANT NOTE ON THE DATA STRUCTURE
In the output of the ParticleTracker plugin, x and y coordinates are inverted (values in "x" column is actually y values and the values in "y" column are x values). According to the author of the plugin, they cannot change this for maintaining the consistency with the matlab script they have. We will see bias in the radial plot but keep in mind that is 90 degrees rotated.

Exercise: Accessing data in 2D table

If data is in a table format (2D vector, such as the case with our data "ptdata"), then an element in the table could be specified by a form "data[row index, column index]". Both row and column number start from 1 so if you want to get a number at the top-left corner of the table we have just imported, way of specifying that cell is

```
ptdata[1, 1]
```

and one column to the right would be

```
ptdata[1, 2]
```

To specify a column, not only a single cell

```
data[, columnnumber]
```

Where a row number should be specified is now blank. It means that all number applies there, which in turn means all the rows available. Another way to specify a column by name of the column header is also available. If we want to specify a column "Trajectory" in the table,

```
ptdata$Trajectory
```

This means "Column Trajectory in data ptdata". By "dataname + dollar sign ($) + the column header", you could specify a column vector within in the table.

Exercise: Extracting Vectors out of DataFrame

Our aim is to extract x and y coordinates, and calculate the angle of the movement from one time point to the other. For this, we first extract x and y vectors out of the data. Don't be afraid with the name "vector". It simply is a term that means "train of numbers". We first try to extract x and y coordinates of a specific trajectory. In the case below, Trajectory No. 2 will be extracted.

```
t1x <- ptdata[ptdata$Trajectory==2, 4]
t1y <- ptdata[ptdata$Trajectory==2, 5]
```

Two lines of commands are executed individually as you press "return" key. Both lines are doing similar job each for x and y. Translation of these lines into normal word would be something like this:

```
From vector ptdata, copy column index 4 to t1x
for only those rows which have values equal to 2
in column Trajectory.
```

We specified a range of rows by `ptdata\$trajectory==2`. To see what this is doing, try

```
ptdata$Trajectory==2
```

You will then see a sequence of statements with ”True” and ”False” This tests for all the values in the Trajectory column, and returns ”Yes (True)” if the value is 2. This could also be written by column index rather than column header as the following

```
ptdata[, 2]==2
```

The returned values will be the same.

You could check if vectors were extracted successfully by clicking corresponding variable name (`t1x` or `t1y`) in the ”Workspace” tab. Alternatively (which actually is the traditional way in R) you could type

```
t1x
```

in the command line. Content of the vector will be printed out. If the length of data is too long, then you could always use head() function to truncate the print out.

```
head(t1x)
```

Exercise: Getting movement vectors for each time points

We now try to get movement vectors. We need to calculate vectors that are made between the position of dot at frame *t* and the position in the next frame *t+1*. To do so, we extract two vectors, each with one element less than the original coordinate vector and one element shifted in one of the two vectors. This could be done as follows.

```
d1x <- t1x[2:length(t1x)] - t1x[1:length(t1x)-1]
d1y <- t1y[2:length(t1y)] - t1y[1:length(t1y)-1]
```

Colon in the square brackets are for generating numerical sequence as we did in the short introduction.

In the right side assignment, we use t1x vector two times, but calling different ranges. In the first term, the range starts from second element of t1x vector and extends until the end. The second term calls from the beginning of the vector and extends until the second last element.

. . . the same thing could be done in much simpler way by using `diff()` function.

```
d1x <- diff(t1x)
d1y <- diff(t1y)
```

Check that d1x actually is the difference generated from t1x.

Exercise: Getting the direction of movement vectors

We now want to get the direction of each movement vector. This could be done by using the function offered by the package fisheyeR, which you loaded in the beginning of this protocol. This function converts Cartesian coordinate to polar coordinate.

```
tpol1 <- toPolar(d1x, d1y)
```

The returned value tpol1 should have twice the length of d1x or d1y. This is because in the first half, theta values are listed and in the last half, magnitude is listed. Our interest is in the theta values, the angle or direction of the movement vector. Check the values by

```
tpol1
```

Calculate directions of all movement vector

Down to here we tired calculating direction of movement vectors only in a single trajectory. What we want actually is to calculate all the directions of movement occurring in the image sequence to examine if there is any bias in the movement direction. To do so, we could use function `diff()` but there is one problem. Since all data are in a single table and trajectories are in same columns, `diff()` will calculate the movement vector that is made between the last coordinates of trajectory *n* and the coordinates in the first time point of trajectory *n+1*. To avoid this, we prepare `diff()` of trajectory id, and use that vector as a flag for elimination of diff data from the `diff()` of x and y coordinates. We first take an example with a short vectors.

Prepare a sample sequence of x coordinates and get diff.

```
aa <-c(1:10, 101:110)
daa <-diff(aa)
```

The function `c()` constructs a vector by combining arguments so typing aa should output

```
[1]   1   2   3   4   5   6   7   8   9  10 101 102 103 104 105 106 107 108 109
[20] 110
```

Then prepare a sample sequence of trajectory id.

```
bb <- c(rep(c(1), 10), rep(c(2), 10))
dbb <- diff(bb)
```

Function `rep()` will repeat the numerical sequence in the first argument up to the number given in the second argument. So typing bb should return

```
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

diff of this vector then should be

```
[1] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
```

As such, we obtain a sequence dbb that flags where data should be removed from diff of aa, daa. We then pass a condition for the daa in the following way.

```
daac <- daa[dbb == 0]
```

We apply such data processing to our real data.

```
dtx <- diff(ptdata$x)
dty <- diff(ptdata$y)
dtraj <- diff(ptdata$Trajectory)
dtxc <- dtx[dtraj == 0]
dtyc <- dty[dtraj == 0]
pol = toPolar(dtxc, dtyc)
```

`ptdata\$x` will return the column with the header "x" from the table in `ptdata`. In the same way, we could isolate y coordinates as a vector in the second line. Other lines are same procedure as we tested with the examples. We now have a vector *pol*, which stores polar coordinates of movement vectors.

Since pol is double the length of x or y coordinate columns, we need to isolate only angles.

```
angledata <- pol[1:(length(pol)/2)]
outdata <- data.frame(angledata)
```

We now have data in *angledata*. We also could get magnitude of movement vectors, which is the displacement per frame (velocity) by extracting the latter half of *pol*. The second line stores in data frame that will be useful only to look at the data and is optional.

To analyze data, one best way to visualize the bias is plotting a histogram.

```
#plotting general histogram
bins <- seq(-pi, pi, pi/50)
hh <- hist(angledata, breaks=bins)
```

We first prepare a vector that defines breaks for the histogram bins. This is done by the function `seq(start, end, increment)`.

Then use the function `hist(data, breaks=bins)` is used to get the histogram values. You will then see a plot in the plot tab in the right-bottom panel. To have more information in the plot, or to adjust plotting parameters, see help by

```
help(hist)
```

To further improve the plot, we could see the distribution by using radial plotting function. We first need to load a package required for radial plotting.

```
#plotting radial plots
library(plotrix)
```

There are many functions in various packages that allow polar plots (or ”radial plot” or ”circular plot”). For small number of sampling any of them works well but radial plot function in `plotrix` seems to be better for data with a large sampling number.

We first extract histogram counts from `hh`

```
# prepare index
hhcounts <- hh$counts
```

Then prepare index for plotting in the radial plot. Unlike histogram, where breaks define the starting point of a bin and the bin extends until the next break, we have single point for representing data. For this, we need to adjust the position by half the width of bin.

```
radpos <- hh$mids
labpos = seq(-pi, 3/4*pi, by=pi/4)
radlabels <- as.character(format(labpos, digits=2))
```

`hh$mids` is a vector with values at the midpoints of bin breaks. Note that if we calculate the same values from break positions, it would be something like (you don't need to do this)

```
radpos <- hh$breaks[1:length(hh$breaks)-1] + diff(hh$breaks)/2
```

. See help of ”hist” for more information on ”mids”. function `as.charactor()` converts numbers that is formatted by `format()` for limiting digits to character. Using these vectors prepared for plotting, we finally do the radial.plot command.

```
radial.plot(hh$counts,radpos,
            rp.type="p",
            main="EB1 directionality",
            line.col="blue",
            labels=radlabels,
            label.pos=labpos,
            radial.lim=c(0, 60),
            mar=c(2, 2, 6, 2))
```

Note that for a command with many options, one could separate them in multiple lines as long as the outer parenthesis is not closed. Maximum value must be adjusted so that the plots fit inside.

Circular Statistics

Circular statistics value could be easily calculated by using the package ”CircStats” to know mean direction, circular dispersion. For mean direction

```
circ.mean(angledata)
```

will prints out the mean. Note that we are dealing with circular data, and the mean direction might not mean anything if the data is distributed uniformly. To know how data is dispersed, use the following command:

```
circ.disp(angledata)
```

This will print several outputs:

```
     n       r      rbar      var
1 3340 459.527 0.1375829 0.862417
```

The value `var` is the circular dispersion of the data, that ranges between 0 and 1. If closer to 0, then it means that data are concentrated in the mean direction, and if closer to 1, data are dispersed to a large degree. One could compare the angular dispersion value of ”early” and ”late” data to see if there is any convergence in the movement direction in the late phase. To test the uniformity (randomness) of direction, Kuiper’s test could be used:

```
kuiper(angledata, alpha=0.05)
```

We used a significance level of 0.05, and out put could be

```
       Kuiper's Test of Uniformity

Test Statistic: 6.9836
Level 0.05 Critical Value: 1.747
Reject Null Hypothesis
```

and this example case tells you that the direction is none-uniform.

Another test available in the package is Rao’s spacing test for the uniformity of data, and is as follows:

```
rao.spacing(angledata)
```

then the output could be

```
       Rao's Spacing Test of Uniformity

Test Statistic = 137.0229
0.01 < P-value < 0.05
```

So if you take a significance level of 0.05, null hypothesis (data is uniformly distributed around circle) is rejected.

If you see that the movement is mostly unidirectional, then we could treat the movement using the von Mises circular statistics. Mean value μ and concentration parameter, or degree of bias in the mean direction, κ could be calculated as follows, and place the values in the plot using text command.

```
text(-100, 80, paste("myu: ",as.character(vm.ml(angledata)$mu)))
text(-100, 70, paste("kappa: ",as.character(vm.ml(angledata)$kappa)))
```

function `paste(,)` concatenates characters.

TIP If the movement is bidirectional (for example, if the movement is biased in both 0 and 3.14 radian) then the concentration parameter κ will not work well since it assumes that the bias is unidirectional. In such a case, one way is to multiply the angle data by two. This operation will bring the bidirectional movement to unidirectional and analysis of kappa value becomes valid.

Putting all into a script

If you are successful in plotting, then you could write a script that does all these in one shot. Useful tool for doing this is the ”History” tab in the top-right panel.

First, create a new script file using the menu command.

`[File > New > R Script]`

You will then see a blank script in a tab in the top-left panel. You could write all the commands by you hand, but more efficient if you use your ”History”. Click ”History” tab in the top-right panel. This tab shows you all the command history. At the top of the ”History”, there are several tools available. One of them is **To Source**, and this could be used to transfer lines you select to the script in the left side.

Scroll back to the position where you did the data importing using read.csv.

```
ptdata <- read.csv("Z:/11EMBOcourse/trials/PTresults.csv")
```

Select the line, and then click ”To Source”. Check the script. You will see that the line is now in the script.

To test using the script, let's remove the data using commandline

```
remove(ptdata)
```

This command removes the ptdata from the workspace. Check the workspace and confirm that the data `ptdata` is not there anymore. Now, go back to the script, and select the line that was transferred from the history. Then click ”Run” button in the top bar of the script tab. This will execute the selected command in the script file.

Check that the data `ptdata` is now in the workspace again. We tested ”To Source” and ”Run” buttons with a single line, but we could also ”To Source” and ”Run” multiple lines selected.

Go back to the history, cherry pick the commands that are necessary and transfer them into the source file. Be careful about the order of commands.

After reconstructing the sequence of commands, you could save the script as a file. File name is automatically appended with extension ”.R”. After the file is saved, you could execute all lines in the script by ”Source” button in the ”script” tab.

3 Protocol 3. Particle Image Velocitometry plugin to Calculate Vector Field

As you are now experienced with R, I only give rough draft of the analysis procedure with data coming from other processing. Particle Image Velocitometry is another way of measuring movement besides particle tracking. The PIV plugin could be downloaded from `https://sites.google.com/site/qingzongtseng/piv`. Note that there is already another plugin in Fiji with the same name ”PIV”. The one we download is more suited for our analysis. Output of the data already contains direction of vectors. We could then import the file in R, extract direction data and plot it in a similar way as it was shown in the Protocol 2.
Column headers in the data file is as follows.

x	y	ux1	uy1	mag1	ang1	p1	ux2	uy2	mag2	ang2	p2	ux0	uy0	mag0	flag

. . . so all we need is to get data from column index 6.

4 Protocol 4. Direction from a reference point

For those who have done everything down to here and have nothing to do: In case of the sequence ”eb1_8b.tif”, eb1 movement is clearly directed outward from the cell center. To quantify this in a better manner than taking the direction as the angle against image, each movement vector against cell centroid could be calculated, so that the relative movement within cell could be analyzed.

If a movement vector we calculated in the protocol 1 and 3 is $\vec{v_m}$ and the position of cell centroid is $\vec{v_c}$, then the angle made between these two vectors could simply be calculated by using dot product and arccos:

$$cos(\theta) = \frac{\vec{v_c} \cdot \vec{v_m}}{|\vec{v_c}||\vec{v_m}|}$$

$$\theta = acos(\frac{\vec{v_c} \cdot \vec{v_m}}{|\vec{v_c}||\vec{v_m}|})$$