NENS 230

Assignment 3: Loops and control flow

Logicals and loops with looped DNA

Due date: Tuesday, 14 October 2014

Goals

- Use `for` loops to iterate though datasets.
- Control program flow via logical `if` statements.
- Gain exposure to basic signal and image processing in Matlab.

Introduction

This assignment is based on the following paper: Extreme Bendability of DNA Less than 100 Base Pairs Long Revealed by SingleMolecule Cyclization by Vafabakhsh and Ha and published in *Science* in 2012. It is only three pages long, so take a look, but you dont have to understand all the details to complete the problem set. We've included the paper in this week's assignment materials.

The authors observed very short pieces of DNA folding into loops. This is exciting because DNA must bend very tightly during gene expression and other processes in living cells, but biophysics experiments on isolated DNA strands predict that they become very stiff at lengths shorter than 150 bp (base pairs). Therefore tight bends should be energetically costly. Here, the authors use an elegant singlemolecule imaging technique to help resolve the conundrum by showing that bending can in fact occur with only weak dependence on lengths under 100 bp.

The authors tether 90 bp strands of DNA to a substrate, and label each end with a different fluorophore. Each strand can either be unlooped or looped, depending on whether the two overhanging ends happen to bind together. One fluorophore (the donor) is excited by blue photons, and emits green photons. The other (the acceptor) is excited by green photons, and emits red photons. These fluorophores were chosen so that when they are brought close together by the action of the DNA loop closing, they will exhibit a phenomenon called Förster (Fluorescence) resonance energy transfer, or **FRET**. FRET occurs when the donor fluorophore is excited by blue light, but instead of emitting a green photon, its energy gets transferred to the acceptor fluorophore, which then emits a red photon.

To summarize, when mostly green light is emitted (directly from the donor), the two molecules must be far apart, implying that the DNA is unlooped. But when mostly red light is emitted (indirectly from the acceptor, via FRET from the donor), the two molecules must be close together, and the DNA is looped. Figure 1, modified from the paper, diagrams this process.

In the paper, the authors characterize the rate of DNA looping in two ways. First, they image a plate of DNA molecules at various time points and build a histogram of the FRET intensity, then extract the

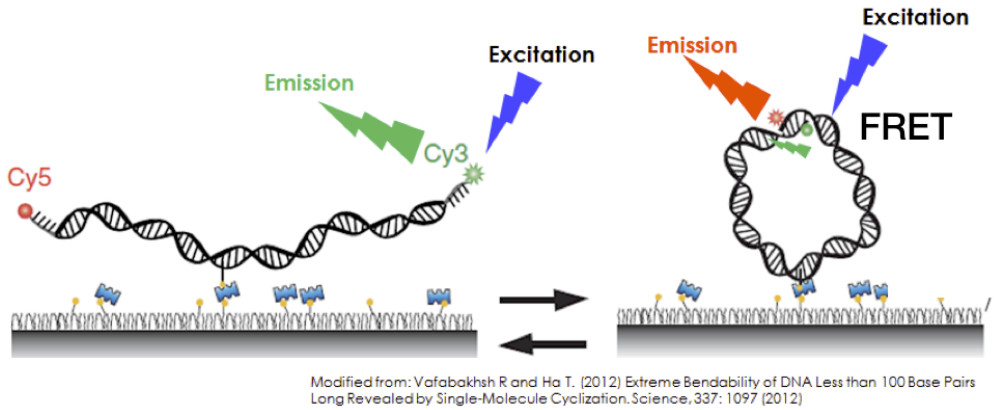

Figure 1: Diagram of the FRET emission of DNA loops in Vafabakhsh and Ha, 2012

fraction of the entire population at high FRET during each time point (Figure 1 in the paper). Second, they watch single molecules over long periods of time, and quantify the rate of transitions between unlooped and looped states (Figures 3a and b). Well start with the second approach, and then move on to the first.

Disclaimer: All the data in this problem set is fabricated and for instruction only. It is based on the actual kinetics reported in the paper, but is not real data!

1 Single molecule FRET analysis

A) Load the file `singlemolecule.mat`. Take a look at the variables:

- F_D - Donor fluorescence trace. This is how much green light was captured from a single molecule during a recording session.
- F_A - Acceptor fluorescence trace. This is how much red light was captured from the same molecule during a recording session.
- t - Time vector in seconds for the single molecule data. The sampling rate is 1Hz, or every second.

Plot a graph of the donor trace in green and the acceptor trace in red against the time vector (similar to panel 3A of the paper). Make sure to label your axes (with units where appropriate), and add a legend using the `legend` command. Remember, you can figure out how the command works by running `help legend`.

B) Calculate the FRET ratio as the following quantity:

$$\frac{F_A}{F_A + F_D}$$

Where F_A is the acceptor trace and F_D is the donor trace. Plot this ratio against time, in your favorite color other than blue.

C) We are now going to threshold this ratio to form a `logical` vector which has a 1 when the ratio is above the threshold, and 0 otherwise. You can pick a reasonable threshold from inspecting your plot of the FRET ratio over time. In your code, name your threshold variable `threshold` and store the logical vector in a variable named `loopedDNA`.

D) Compute the probability of being looped. This is simply the amount of time spent in the looped state divided by the total time of the experiment. It should be easy to compute this using your `loopedDNA` vector. Print this probability to the screen using `fprintf`, printing exactly 3 places after the decimal point. (One easy sanity check you can do is to make sure your probability is between 0 and 1!)

E) Now for the hard part. Let's assume the unlooped to looped transition is a first order reaction, and so we can compute the rate of looping (k_L). You can do this by dividing the total number of unlooped to looped transitions by the time spent in the unlooped state. This will give us a rate constant with units of $\frac{1}{s}$, or Hz, and can be thought of as the rate of a unlooped strands becoming looped in some time window. Similarly, we can find the rate of unlooping (k_U) by dividing the total number of looped to unlooped transitions by the total time spent in the looped state.

Here are some more detailed instructions on how to do this: To find the number of unlooped to looped transitions, and vice versa, we will use a `for` loop. Start by pre-allocating variables `lowToHigh` and `highToLow` to store the number of transitions of each type (these will contain a single number). Loop through the `loopedDNA` vector. On each iteration, use an `if` statement to check for a transition when the current element in `loopedDNA` is `false` but the next one is true, this is a `lowToHigh` transition, so increment the `lowToHigh` variable. Using the same `if` block (hint: use `elseif`), also check for high to low transitions and similarly increment the `highToLow` variable. *Bonus: This can also be done without a loop, using the* `diff` *command. See if you can learn how!*

Now that you have counted the number transitions, just divide the number of `lowToHigh` transitions by the amount of time spent in the low state (which you can get by taking the sum of `1-loopedDNA`, since each sample covers 1 second). This gives you the probability per unit time that an open stand of DNA will loop. Conversely, k_U is given by the sum of `highToLow` transitions, divided by the amount of time in the looped state.

The overall time constant for the reaction is given by: $\tau = \frac{1}{k_L + k_U}$. Print the values of k_L, k_U, and τ to the screen using `fprintf`. Make sure to print out what the units are for each number quantity as well.

2 Population FRET imaging analysis

A) There are eight images provided, labeled `img1.tif` through `img8.tif`. Each image shows a plate with about 200 single molecules of DNA. Green dots represent unlooped pieces of DNA and red represents looped DNA. The eight images are of the same field of view, but taken at different points in time. The time points are not stored, so first create a time vector to store the times when each picture was taken as follows: `It = [0 1 4 11 18 28 40 75];`

B) Before we get to the `for` loop, let's get a better sense of what the data looks like. From the command line (you don't need to include this in your script), take a look at some of the images by first loading them into Matlab using the `imread` command and then viewing them with `image` or `imagesc`. Look at the `help` for each so you understand the difference between these very useful commands. Additionally, run the `whos` command and note how the data is being stored (it is stored as `uint8`, which stands for an 8-bit unsigned integer). With only 8 bits, we can store `integers` from 0 to $2^8 - 1$, or 0 to 255. We will get around this by converting the data to the `double` format, which allows us to store many more values (including decimals) - more on this later.

C) You will be looping through the set of images and extracting data from each. However, it's generally good practice to build and test loop code using only one or two passes through the loop, i.e, one or two images. This makes it much easier to test and debug. After you're convinced the code works, change the `for` loop to include all 8 images.

You want to create a `for` loop that runs from 1 to 8, corresponding to the 8 images. For each image, do the following:

1. Load the image file into a variable named `img`. This variable will get rewritten on each iteration of the loop by the current image to process. To load the image, you may have to use the `num2str` or `sprintf` command to turn the numerical index of the current iteration into a string, which you can add to the filename you pass to `imread`.

2. Convert the data to the `double` format - there is a function in Matlab conveniently called `double` that will do this for you. (See section B above for an explanation of why we want to do this)

3. Extract the red and green channels from the image. The data is stored as a 3D array where the red, green, and blue color channels are stored in the first, second, and third position of the third dimension, respectively. (These are often called "pages".) You can slice out the red and green channels using the colon syntax we discussed in week two.

4. Calculate the FRET ratio for the whole image. First, you can either sum or average all of the values for each channel, and then you can compute the ratio using the formula used above: $\frac{F_A}{F_A+F_D}$ You should have a single number representing the ratio for that image. Here, as above, F_A corresponds to the red channel (Acceptor) and F_D corresponds to the green channel (Donor). Store the FRET ratio for each image as a value in an array.

D) Now that we have calculated the FRET ratio for each image, we are going to visualize our results. Make a plot of the FRET ratio over time using the time vector `Id` described in (2A). The FRET ratio is the one you just computed using the `for` loop. Additionally, you should also add a horizontal line indicating the steady state value. This value is just the value that the curve seems to be settling at, which you can read off the graph manually. You can add this to the plot by just picking two different x-values that span the time range and then plotting them with the value you chose. For example, `plot([0 10], [5 5])` will create a plot of a horizontal line extending from 0 to 10 on the x-axis with a constant value of 5 on the y-axis.

As always, make sure to label axes with units where appropriate, and add a legends and titles. It should be reasonably clear that the curve you plotted is trending towards settling at the steady state value (straight line) that you plot.

Submission

When you are finished, publish your work by running the command: `publish('assignment3.m', 'pdf')` from the command line. This should generate all of the figures and along with the code and comments from the script put them in a PDF document in a folder named `html`. Email your script (original code) and the generated PDF document to nens230-aut1415-staff@lists.stanford.edu to submit your assignment, and please include the tag `[Assignment 3]` in the subject line.