Analysis of Single Moculecule Photobleaching

Keegan Hines

April 24, 2013

Introductory

This tutorial is a walkthrough of the analysis functions that accompany the paper *Inferring Subunit Stoichiometry from Single Molecule Photobleaching*. Since these functions were written using the language R, this document provides a guide for importing data into R and navigating the R environment as well as descriptions and examples of each of the functions. Users are encouraged to import their own data and use these methods for analysis.

Working in R and loading data

Open up R and have a look around. In particular, make sure that all the files relating to photobleaching analysis are in your current working directory. The function **getwd()** can be used to display the current working directory and we can list the contents of the current directory with the function **dir()**. If the current directory is not correct, use **setwd()** or use the menu options `Misc -> Change Working Directory`.

```
> getwd()

## [1] "/Users/keegan/supplemental"

> dir()

##  [1] "#tutorial.Rnw#"        "bleaching_functions.R"
##  [3] "data1.csv"             "figure"
##  [5] "tutorial.aux"          "tutorial.log"
##  [7] "tutorial.pdf"          "tutorial.Rnw"
##  [9] "tutorial.Rnw~"         "tutorial.tex"
```

My directory has several files named `tutorial.***` which are related to compiling this document (and also a directory called `figure`) that will not be in your directory. Aside from those, your directory should have all the files listed above. In particular, make sure there is a file `bleaching_functions.R`, which contains the functions we'll use, as well as a `.csv` file, which contains example data.

Let's get started. We'll read in everything in the `bleaching_functions.R` file and verify that we now have new functions we can access.

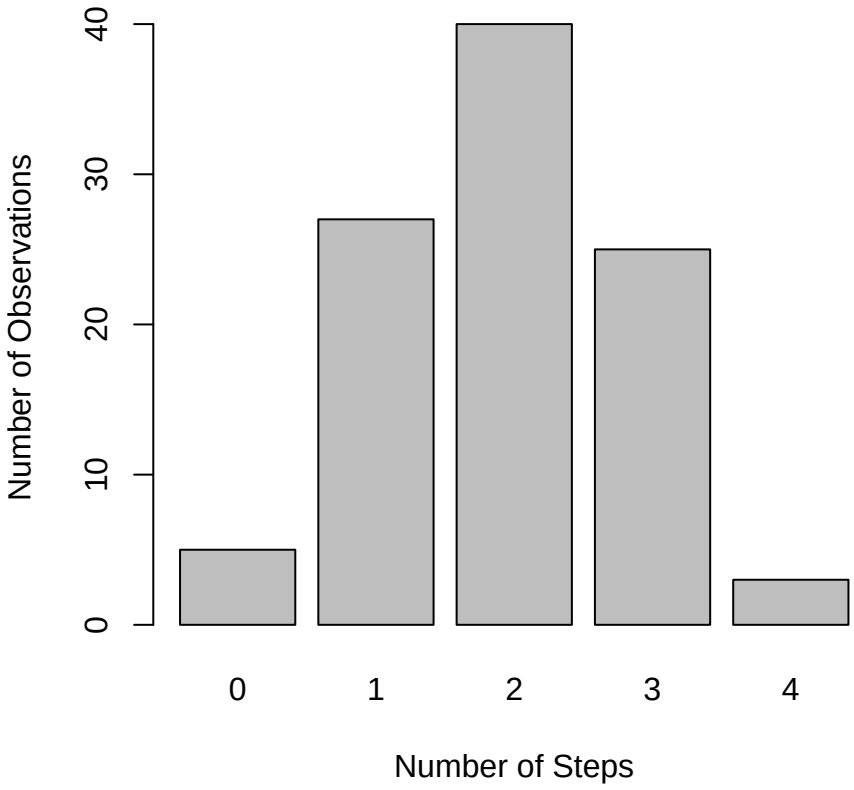
```
> source("bleaching_functions.R")
> ls(1)

## [1] "confidence"     "estimate.theta" "gamma"
```

Suppose I have measured some single molecule photobleaching events and have stored all the counts of bleaching steps in a text file. There is a file `data1.csv` in this directory that will serve as example for how to

import data. This file is a comma separated text file which lists integers which presumably represent counts of bleaching events. Note, these functions will not work with raw photobleaching data, but only with lists of numbers of bleaching steps (ie. lists of integers). Therefore, you will need to format your own data to be similar to the file `data1.csv`. Alternatively, data can also be copy-and-pasted into the console and assigned to a variable. So long as the data is of mode *numeric*, everything should work.

For importing data from a text file, we can create a variable whose contents are read in from the file and then verify that the everything was formatted correctly. We can do a quick visual check of the data to see that it looks binomially distributed.

```
> imported.data <- as.numeric(read.csv("data1.csv", header = FALSE))
> print(imported.data)

##   [1] 3 1 3 1 2 1 0 2 3 2 1 3 1 2 1 1 1 3 1 0 2 2 3 1 3 1 1 3 0 3 2 2 3 1 2
##  [36] 1 0 3 2 2 3 3 1 1 2 1 1 2 1 3 3 2 1 2 3 1 2 2 3 2 1 2 2 3 2 3 2 1 3 2
##  [71] 2 2 1 2 2 2 2 2 2 4 2 2 3 4 2 2 1 2 3 1 0 4 3 3 2 2 1 2 3 2

> barplot(table(imported.data), xlab = c("Number of Steps"), ylab = c("Number of Observations"))
```



For exploratory purpose, we can also generate random data through simulation, which is what we will do for the remainder of this tutorial.

Analysis Functions

One of functions available to us is called **estimate.theta()**. Recall from the main text of the accompanying paper that the parameter θ is the probability of a particular binary event happening. This parameter plays an important role in a binomial distribution, Bn(n,θ), which is the probability distribution of observing any number of events, given that n are possible and each occurs with probability θ. Ultimately, we need to estimate n and θ from some observations. In the paper, it was noted that the estimate of θ depends on an assumption about n such that as n increases, the estimate of θ will decrease. The function **estimate.theta()** provides a point estimate of θ with respect to a particular n. For example, let's simulate some binomially distributed observations and then use those to estimate the true θ.

```
> simulated.data <- rbinom(100, 4, 0.5)  #100 observations from Bn(4,.5) model
> table(simulated.data)

## simulated.data
##  0  1  2  3  4
##  6 21 43 20 10

> estimate.theta(simulated.data)

## [1] "For 4 subunits, best estimate of theta is 0.52"
```

By default, this function finds the largest observation and assumes that to be n. However, we can specify that we wish to estimate θ with respect to any potential n.

```
> estimate.theta(simulated.data, n = 5)

## [1] "For 5 subunits, best estimate of theta is 0.41"
```

It is pointed out in the main text that estimating the true n from such observations can be problematic. As we just saw, the optimal estimate of θ will shift downward as we consider larger n. In some instances, the result will be that many potential n can fit some data identically well. Due to this, methods were developed to quantify confidence when anaylzing such observtions. The two methods of quantification are described in equations [4] and [9] in the main text and are implemented in the function **confidence()**.

This function requires only one argument, the name of the variable containing the data. Alternatively, we can supply a call to **rbinom()** which will simulate random data, and allow us to more easily examine the effects of the parameters on estimation confidence.

```
> confidence(simulated.data)

## [1] "Number of subunits is estimated to be 4 with confidence of 0.7009"

> confidence(rbinom(100, 4, 0.5))

## [1] "Number of subunits is estimated to be 4 with confidence of 0.5848"
```

For example, increasing θ results in increased confidence, as we might have expected. Additionally, larger n results in less confidence.

```
> # Notice effect of increasing theta
> confidence(rbinom(100, 4, 0.6))

## [1] "Number of subunits is estimated to be 4 with confidence of 0.9061"
```

```
> confidence(rbinom(100, 4, 0.7))

## [1] "Number of subunits is estimated to be 4 with confidence of 0.9968"

> # Notice effect of increasing n
> confidence(rbinom(100, 4, 0.7))

## [1] "Number of subunits is estimated to be 4 with confidence of 0.9992"

> confidence(rbinom(100, 8, 0.7))

## [1] "Number of subunits is estimated to be 8 with confidence of 0.7056"
```

I mentioned that the function **confidence()** can implement one of two calculations. By default, this function computes the full Bayesian estimate of confidence which is described by equation [9] of the main text. In the paper, I argue that this method is the most conservative and appropriate way to estimate confidence. Nonetheless, the simpler calculation of equation [4] can also be implemented by including the argument `Bayes=FALSE`.

```
> confidence(simulated.data)

## [1] "Number of subunits is estimated to be 4 with confidence of 0.7009"

> confidence(simulated.data, Bayes = FALSE)

## [1] "Number of subunits is estimated to be 4 with confidence of 0.7043 ."
```

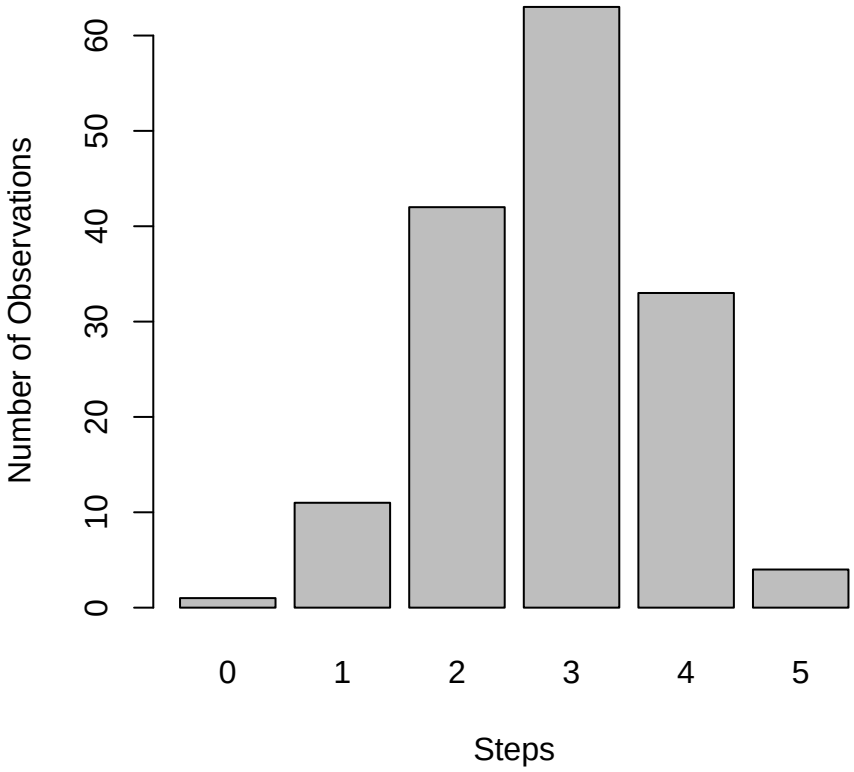
In general, the non-Bayes estimate will be an overestimate of confidence. However, for small sample sizes (or low θ), it is not impossible for the Bayesian estimate to be larger. The reason for this is that the Bayesian estimator takes into the account the full variance in the observations, whereas the simpler method just assumes that θ is known with perfect precision and does not consider the actual data. As a result, the Bayesian estimator is more sensitive to sampling variance in small sample sizes. Additionally, as N increases, the two methods will become equivalent.

```
> confidence(rbinom(350, 4, 0.6))

## [1] "Number of subunits is estimated to be 4 with confidence of 0.9996"

> confidence(rbinom(350, 4, 0.6), Bayes = FALSE)

## [1] "Number of subunits is estimated to be 4 with confidence of 1 ."
```

The final function can be used to address whether certain observations might be artifacts. As described in equation [10] of the main text, the parameter γ is an estimate of whether the largest number of observed n occur with anomalously low prevalence. For example, let's suppose that the true n is four and that a data collection algorithm resulted in a small number of artifactual observations larger than four. We can visually inspect the data and notice that the distribution looks inconsistent. The function **gamma()** provides an estimate of observing a similar distribution under the null hypothesis that n is equal to the largest number of observations. Similar to a p-value, a low value for γ is evidence against the null hypothesis and thus we might exclude all observations larger than four as artifacts.

```
> messy.data <- c(rbinom(150, 4, 0.7), 5, 5, 5, 5)  # We have 150 observations from a
> # binomial distribution with n=4, and also several anomalous observations
```

```
> # of size 5.
> barplot(table(messy.data), xlab = c("Steps"), ylab = c("Number of Observations"))
```




```
> # Data looks strange. It seems we should conclude n=5, but maybe the
> # events of size 5 are artifacts.
> gamma(messy.data, Bayes = FALSE)

## [1] 0.05067
```

In this instance, γ is small and therefore we might reject the null hypothesis, depending on our threshold. If we remove all observations of size 5 from the data set, we can ask again whether the resulting data are anomalous with respect to how many events of size n are observed.

```
> gamma(messy.data[messy.data < 5], Bayes = FALSE)

## [1] 0.4077

> confidence(messy.data[messy.data < 5], Bayes = FALSE)

## [1] "Number of subunits is estimated to be 4 with confidence of 0.9997 ."
```

This estimate of γ is large, which indicates that the distribution is not atypical. Therefore, we might conclude that n=4 and it turns out that we can make that claim with high confidence. This function also has a Bayesian version which takes into account the full uncertainty in θ and is generally more conservative.

```
> gamma(messy.data, Bayes = FALSE)

## [1] 0.05067

> gamma(messy.data)  #Bayesian method is default

## [1] 0.06548
```