Curve Fitting in R

Aaron Taing

December 3, 2017

Introduction

In my previous post, I discussed the use of ggplot2 in visualizing scientific data. I was concerned with the ability to represent data sets in a clean format. However, sometimes we are also interested in extracting quantitative results from the data along with the qualitative representation. For example, perhaps we perform an experiment measuring cell death (or survival) against the concentration of some toxic compound. We may first be interested in fitting a curve to the observed data points. We may then desire to extra an LD-50 value (the dose amount that kills 50% of the population) from the approximated curve.

Thus, this post will deal with fitting curves to observed data. We will explore the example specified above and others. I say "we" because this post will include code chunks that will make the results displayed here completely reproducible so that you the reader will be able to follow along.

Preparing the Data

The first thing that we need to do is to load the packages that I will be using. Most of the data sets we will use are from the package **datasets**, which contains various publicly available datasets. The package **dplyr** will allow us to manipulate the data if necessary.

```
# loading packages
library(datasets) # public data sets
library(dplyr) # to manipulate data
```

The second thing we will do in preparation is to form various data frames from the data sets that we will use. The names I have us assign to each data frame will be explained in the next section. One of the data sets used contains values from an experiment in which I was an assistant [1]. Unfortunately, our github repositories are private since they are for a class, so I will have to type out the data set manually.

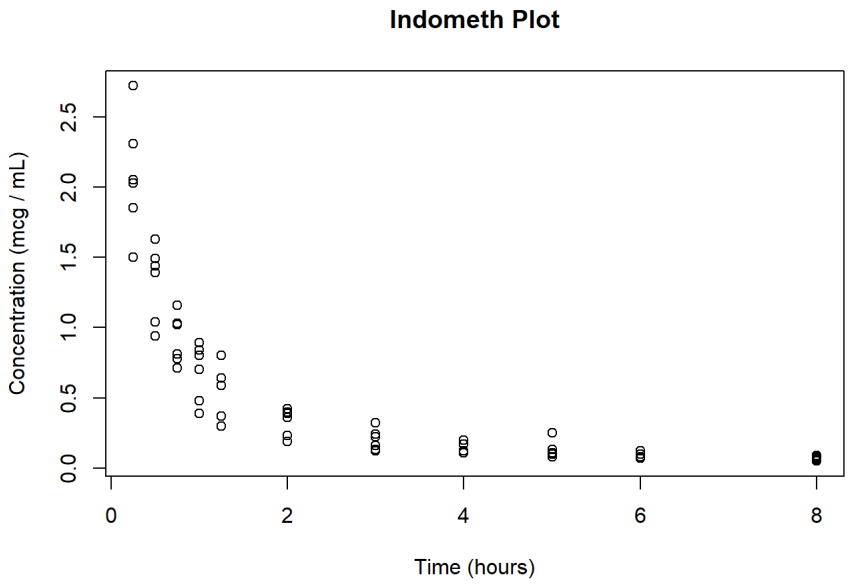
```
dat.lin <- Formaldehyde # from datasets package
dat.exp <- Indometh # from datasets package
dat.log <- Puromycin # from datasets package

# my own data
conc <- c(-4, -3, -2, -1, 0, 1) # log of concentration of Triton-X added
surv <- c(103.0992, 109.4542, 89.91965, 18.48064,
               -8.50464, -3.58969) # percent cell survival
dat.sig <- data.frame(conc, surv)
```

Preliminary Inspection of the Data

Before trying to fit curves to each data set, we need to take a look at each of them to decide what sort of curves can fit them. We will look at very basic plots of each data set. Plots were coded with the help of the base R graphics cheat sheet [2].

The first data set we will look at is **Indometh**. This data set contains values from an experiment that measured the blood concentration of indometacin (**conc**) over time (**time**) in various subjects [3].

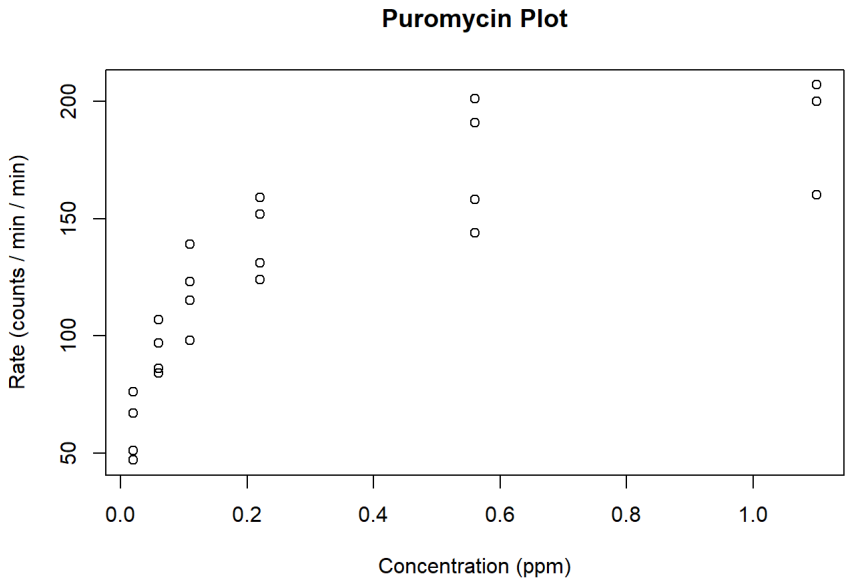
```
plot(dat.exp$conc ~ dat.exp$time, # plotting concentration vs time
     main = "Indometh Plot",
     xlab = "Time (hours)",
     ylab = "Concentration (mcg / mL)") # mcg = micrograms
```



The points appear to follow a trend of exponential decay. Despite having variable subjects, the data points are close enough that we can probably fit a trend line to all of them at once. Therefore in a later section, we will fit an exponential curve to this data set, hence the data frame name **dat.exp**.

The second data set is **Puromycin**. In the experiment that produced this data set, the reaction rate (**rate**) was measured against substrate concentration (**conc**) in cells that were either treated or untreated with Puromycin [4].

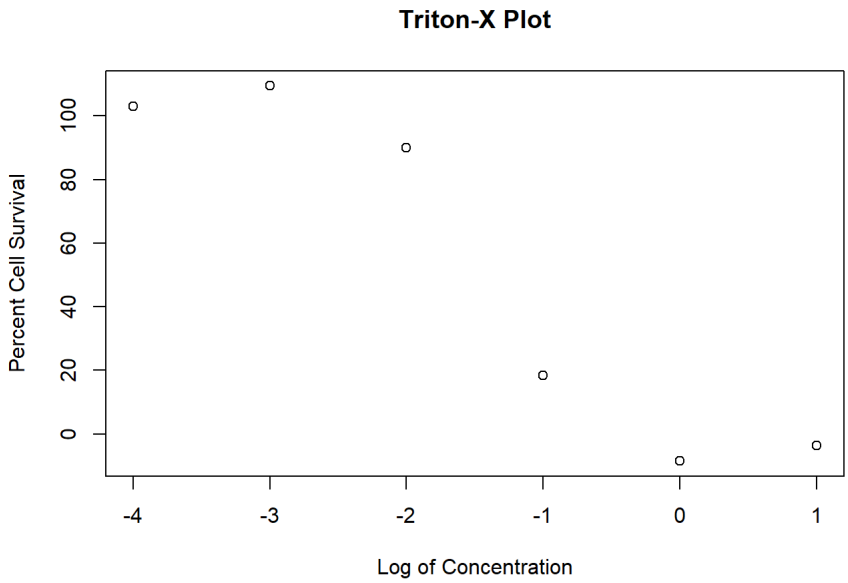
```
plot(dat.log$rate ~ dat.log$conc,
     main = "Puromycin Plot",
     xlab = "Concentration (ppm)", # ppm = parts per million
     ylab = "Rate (counts / min / min)")
```



The points appear to follow a trend of logarithmic growth. The difference between rates in the treated and untreated cells seems to be quite different. Thus, we will fit two logarithmic curves to this data set (**dat.log**).

The final data set is the one I had to manually enter. In this experiment, HEK cells were treated with Triton-X, which is toxic to cells due to its ability to disrupt cell membranes. The percent of cells surviving the treatment (**surv**) was measured against the concentration of Triton-X added (**conc**).

```
plot(dat.sig$surv ~ dat.sig$conc,
     main = "Triton-X Plot",
     xlab = "Log of Concentration",
     ylab = "Percent Cell Survival")
```



The points for this last data set appear to follow a logistic or sigmoidal curve [5], which is why we have named the data frame **dat.sig**.

Exponential Curve Fitting

We saw that the data set **Indometh** could likely be fitted with an exponential curve. An exponential curve has the general form of $a * e^{bx}$. We need to use R to estimate values for a and b. We can accomplish this with the function **nls()** [6]. In this function's paramters, we will tell it to use the general exponential curve formula above, and we will give it starting values for a and b as a rough guess. By inspecting the basic plot previously created, we can guess a starting value of about 2 for a and -1 for b.

```
fit.exp <- nls(conc ~ a * exp(b * time), # general exponential formula
               data = dat.exp,
               start = c(a = 2, b = -1)) # guessed starting values
summary(fit.exp)
```

```
##
## Formula: conc ~ a * exp(b * time)
##
## Parameters:
##   Estimate Std. Error t value Pr(>|t|)
## a  2.77705    0.15379   18.06   <2e-16 ***
## b -1.35037    0.09938  -13.59   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2001 on 64 degrees of freedom
##
## Number of iterations to convergence: 7
## Achieved convergence tolerance: 9.042e-06
```

We can see from the output of **summary(fit.exp)** that the estimated a and b values are 2.77705 and -1.35037 respectively. Now we have a specific formula to use with those predicted coefficient values. We can then use the function **predict()** to generate predicted y-values given input x-values using the approximated formula [7]. We enter as parameters in this function the results of the **nls()** function and a data frame containing the x-values from which we will predict y-values. The data frame parameter is called by **newdata =** and must use the same variable name as x-values in the data set [8]. Let us take a look at some predicted y-values.

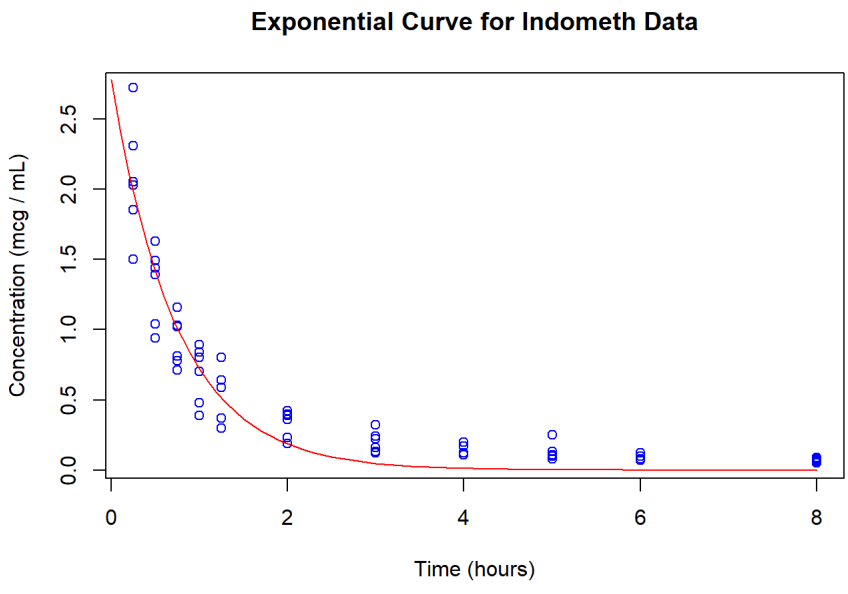
```
predict(fit.exp, # calling predict using the result of nls()
        newdata = data.frame(time = seq(0, 8, by = 1)))
```

```
## [1] 2.777053e+00 7.196543e-01 1.864935e-01 4.832852e-02 1.252401e-02
## [6] 3.245512e-03 8.410523e-04 2.179530e-04 5.648103e-05
```

```
# used same x-variable "time" as in dat.exp
```

Note that the paramter **by =** sets how many predicted y-values are generated by setting an interval size. We will plot this trend line by calling the function **lines()** to plot the predicted y-values against our desired x-values. The **time** variable in **dat.exp** goes from 0 to 8, so we will plot from 0 to 8 with a predicted y-value generated for every step of 0.1 (**by = 0.1**).

```
plot(dat.exp$conc ~ dat.exp$time,
     main = "Exponential Curve for Indometh Data",
     ylab = "Concentration (mcg / mL)",
     xlab = "Time (hours)",
     col = "blue") +

  lines(seq(0, 8, by = 0.1), # plotting line from 0 to 8 with interval size 0.1
        predict(fit.exp, # using predict() to generate y-values
                newdata = data.frame(time = seq(0, 8, by = 0.1))),
        col = "red")
```




```
## integer(0)
```

Now we have created a nice trendline for the **Indometh** data set!

Logarithmic Curve Fitting

Logarithmic Curve Fitting

We saw that the data set **Puromycin** can be fitted with 2 logarithmic curves. The general form of a logarithmic equation is $a * log(x) + b$, where log stands for the natural logarithm. First we will split the data set into 2 data sets with **state = treated** and **state = untreated**. Then, as in the previous section, we will use **nls()** to estimate the a and b parameters for the formula. The values are rather more difficult to guess this time, so let us just use a starting value of 1 for both a and b.

```
# splitting the data set into two
dat.log.t <- filter(dat.log, dat.log$state == "treated")
dat.log.u <- filter(dat.log, dat.log$state == "untreated")

# estimating a and b
fit.log.t <- nls(rate ~ a * log(conc) + b, data = dat.log.t,
             start = c(a = 1, b = 1))
fit.log.u <- nls(rate ~ a * log(conc) + b, data = dat.log.u,
             start = c(a = 1, b = 1))

# taking a look at the predicted a and b
summary(fit.log.t)
```

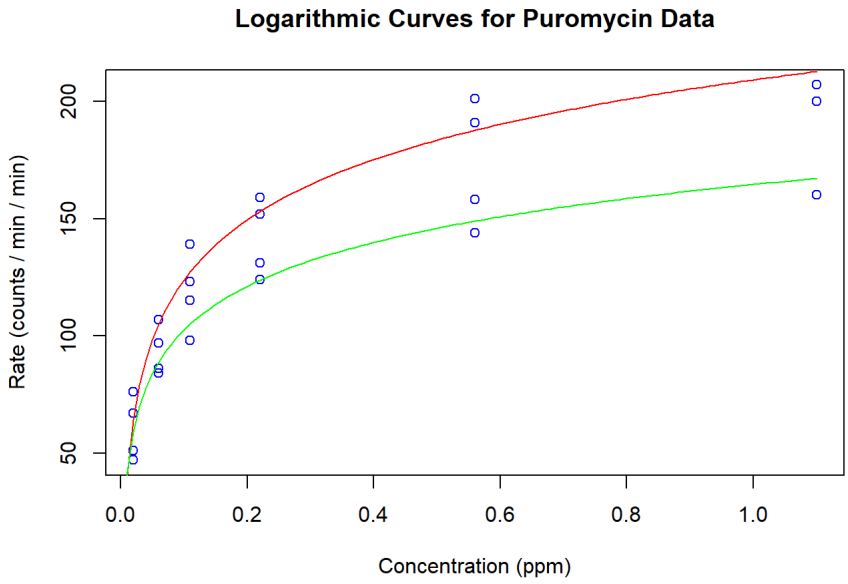
```
##
## Formula: rate ~ a * log(conc) + b
##
## Parameters:
##   Estimate Std. Error t value Pr(>|t|)
## a   37.110      2.229   16.65 1.28e-08 ***
## b  209.194      5.045   41.47 1.59e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.37 on 10 degrees of freedom
##
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 1.274e-08
```

```
summary(fit.log.u)
```

```
##
## Formula: rate ~ a * log(conc) + b
##
## Parameters:
##   Estimate Std. Error t value Pr(>|t|)
## a   26.982      1.805   14.95 1.16e-07 ***
## b  164.588      4.266   38.59 2.62e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.575 on 9 degrees of freedom
##
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 1.411e-08
```

Now with the generated a and b values, we can plot our logarithmic curve. The procedure is exactly the same as with plotting the exponential curve.

```
# generating the plot
plot(dat.log$rate ~ dat.log$conc,
     main = "Logarithmic Curves for Puromycin Data",
     xlab = "Concentration (ppm)",
     ylab = "Rate (counts / min / min)",
     col = "blue") +
  # generating lines using y-values from predict()
  lines(seq(0, 1.1, by = 0.01),
        predict(fit.log.t, newdata = data.frame(conc = seq(0, 1.1, by = 0.01))),
        col = "red") +

  lines(seq(0, 1.1, by = 0.01),
        predict(fit.log.u, newdata = data.frame(conc = seq(0, 1.1, by = 0.01))),
        col = "green")
```



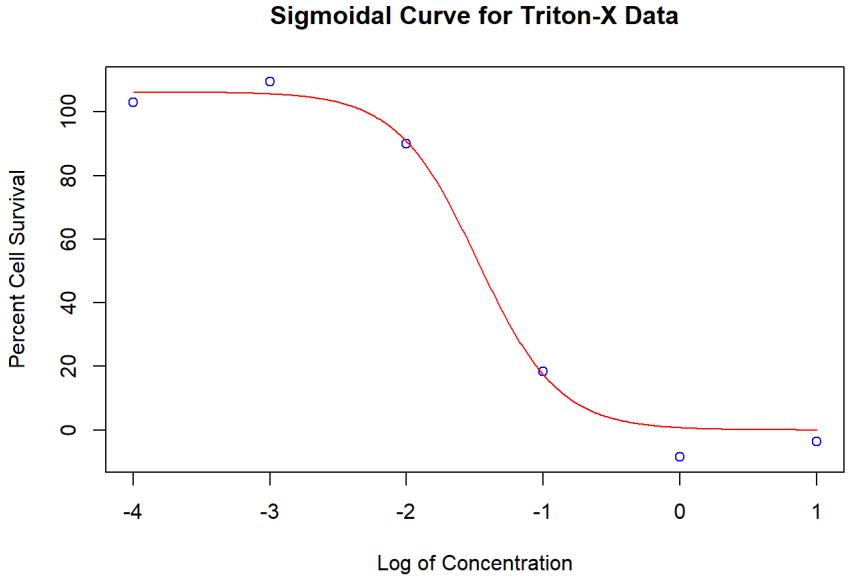

```
## integer(0)
```

Now we also have nice logarithmic trendlines!

Sigmoidal Curve Fitting

Finally we can fit a sigmoidal curve to the data set I entered manually. A sigmoidal curve follows the general form described in [5]. For a sigmoidal fit, R actually provides a model with the function **SSlogis(input, Asym, xmid, scal)** [7]. We can call this function in **nls()** to generate estimated parameters. Then we can use **predict()** to get estimated y-values. Finally we can plot the estimated y-values against chosen x-values. Same procedure as before, all in 1 chunk now:

```
# using SSlogis inside nls() with input = conc (the x-variable in dat.sig)
fit.sig <- nls(surv ~ SSlogis(conc, Asym, xmid, scal), data = dat.sig)

# plotting
plot(dat.sig$surv ~ dat.sig$conc,
     main = "Sigmoidal Curve for Triton-X Data",
     xlab = "Log of Concentration",
     ylab = "Percent Cell Survival",
     col = "blue") +

  # plotting predicted y-values
  lines(seq(-4, 1, by = 0.01),
        predict(fit.sig, newdata = data.frame(conc = seq(-4, 1, by = 0.01))),
        col = "red")
```




```
## integer(0)
```

So now we also have a sigmoidal curve fitted to data. Before we conclude however, there is one last thing we can do. At the very beginning of

this post, I mentioned that we can extract quantitative values from this method in addition to obtaining qualitative results. Throughout this post, we have managed to extract quantitative results through estimated formulas. Each time, we followed a general formula and obtained values for the coefficients or parameters in those formulas. We also saw that we could see the predicted y-values. That brings us to…

Finding the LD-50 value from our Sigmoidal Curve

From the predicted y-values, we can find the LD-50 for Triton-X for HEK cells. In general, the LD-50 value is quite important in biological research.

```
# the predicted y values
y <- predict(fit.sig, newdata = data.frame(conc = seq(-4, 1, by = 0.005)))

# the x values used for generating the y-values
x <- seq(-4, 1, by = 0.005)

# put them into a data frame
dat <- data.frame(x, y)
```

The idea is that since our y-value represents the percent of cells surviving, we can look at the predicted y-values and look at what x-value generates a y-value very close to 50. That x-value would be the LD-50. The data frame generated in that code chunk has 1001 observations, so it is probably not a good idea to try and print it out for us to peruse. Let us cut the data frame to include only observations with y-values of 49 to 51.

```
# truncated data frame
dat.trunc <- filter(dat, dat$y > 49 & dat$y < 51)
print(dat.trunc)
```

```
##         x        y
## 1 -1.455 50.90003
## 2 -1.450 50.45029
## 3 -1.445 50.00093
## 4 -1.440 49.55203
## 5 -1.435 49.10363
```

At x = -1.445, y = 50.00093. This x-value (remember that x is a log of the concentration of Triton-X) seems to be a good estimate of the LD-50 value!

Conclusion

There are of course other general formulas that trendlines can follow, but I hoped to capture common ones that result from natural laws of growth and decay. I hope that you were able to see how convenient curve-fitting can be in R, even without using **ggplot2**! Of course, this can be done in **ggplot2** as well, but it's clear to see that base R can handle curve fitting well.

I hope you enjoyed this post. Good luck to you, the reader, for the upcoming finals!

References

1. Roeise, J., Viability of HEK Cells with Triton X.
2. http://publish.illinois.edu/johnrgallagher/files/2015/10/BaseGraphicsCheatsheet.pdf
3. https://www.rdocumentation.org/packages/datasets/versions/3.4.1/topics/Indometh
4. https://www.rdocumentation.org/packages/datasets/versions/3.4.1/topics/Puromycin
5. http://kyrcha.info/2012/07/08/tutorials-fitting-a-sigmoid-function-in-r/
6. https://stackoverflow.com/questions/15102254/how-do-i-add-different-trend-lines-in-r
7. https://stackoverflow.com/questions/33033176/using-r-to-fit-a-sigmoidal-curve
8. https://stackoverflow.com/questions/15173086/plot-new-values-for-best-fit-nonlinear-curve