<http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm>

# Lecture 7—Monday, September 20, 2010

### Topics

* [The aphid count data set](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#aphid)
* [Poisson model](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#poisson)
  + [Constructing the Poisson log-likelihood](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#constructing)
  + [Graphing the log-likelihood](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#graphing)
  + [Maximizing the log-likelihood numerically](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#maximizing)
  + [Assessing the fit graphically](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#assessing)
* [Negative binomial model](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#negativebinomial)
  + [Constructing the negative binomial log-likelihood](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#NBlogL)
  + [Obtaining the MLE](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#NBMLE)
  + [Assessing the fit of the negative binomial model graphically](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#NBassess)
* [Pearson chi-square goodness of fit test for discrete data](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#pearson)
* [Testing the fit of the Poisson model](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#poissontest)
  + [Parametric goodness of fit test](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#poissonparametric)
  + [Simulation-based goodness of fit test](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#poissonsimulate)
* [Testing the fit of the negative binomial model](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#NBtest)
  + [Parametric goodness of fit test](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#NBparametric)
  + [Simulation-based goodness of fit test](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#NBsimulate)
* [Cited references](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#cited)
* [R code](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#Rcode)

### R functions and commands demonstrated

* [barplot](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#barplot) is the base graphics function for generating bar plots.
* [chisq.test](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#chisqtest) can be used to perform a Monte Carlo goodness of fit test based on the Pearson chi-squared statistic.
* [nlm](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#nlm) is the nonlinear minimization function of R.
* [optim](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#optim) is another numerical optimization function of R. It offers more optimization methods than does nlm.
* [pchisq](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#qchisq) is the cumulative distribution function of the chi-squared distribution.
* [prod](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#prod) calculates the product of all the elements of a vector.
* [qchisq](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#qchisq) is the quantile function of the chi-squared distribution.
* [rep](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#rep) generates a vector whose elements are repeated a specified number of times.
* [sapply](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#sapply) is used to apply a vector individually to elements in a vector returning one value for each component of the vector. The return value of the function is coerced to be a vector or a matrix.

### R function options

* [B](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#Barg)= (argument to chisq.test) can be used to modify the default number of simulations (B=2000) for the simulation-based version of the chi-squared test.
* [hessian](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#NBMLE)= (argument to **nlm**) can be assigned the values TRUE or FALSE (the default). When TRUE the Hessian is reported evaluated at the minimum value obtained by **nlm**.
* [names.arg](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#namesarg)= (argument to barplot) can be used to add labels to the bars.
* [simulate.p.value](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#simulatepvalue)= (argument to **chisq.test**) can be assigned the values TRUE or FALSE. When TRUE a Monte Carlo-based p-value is calculated for the observed Pearson chi-squared statistic.
* [type](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#type)= (argument to plot, points, and lines) controls what is plotted. We used **type='l'** in this exercise to plot a line and type='o' to plot both lines and points.

### The aphid count data set

The data set for today's exercise is shown below. It appears on p. 130 of Krebs, Charles. 1999. *Ecological Methodology*. Menlo Park, CA: Addison-Wesley. He doesn't give a source for it.

#### Table 1—Today's Data

|  |  |
| --- | --- |
| **# of aphids on a stem** | **Number of stems** |
| 0 | 6 |
| 1 | 8 |
| 2 | 9 |
| 3 | 6 |
| 4 | 6 |
| 5 | 2 |
| 6 | 5 |
| 7 | 3 |
| 8 | 1 |
| 9 | 4 |
| **Total** | **50** |

Our goal today is to fit Poisson and negative binomial distributions to these data. Since the Poisson distribution is a one-parameter distribution (with parameter λ) the problem for the Poisson reduces to estimating λ. Typically Poisson regression models include predictors in which the goal is to make λ a function of those predictors. What we're doing today is fitting a Poisson regression model in which the only predictor is an intercept.

The data above are given in summary form. The raw data would consist of a list of stems with a count of how many aphids were found on each stem. The table above groups the stems together based on how many aphids were found. The raw data can be recreated from the grouped data with R's rep function. With rep we can repeat the numbers in the first column—0, 1, 3, …, 9—the exact number of times listed in the second column.

First let's explore how rep works. The first argument to rep is what we want to repeat; the second argument describes how the repetition is supposed to be done.

* If the second argument is a single number, then the first argument is repeated that many times as a unit.

rep(4,10)  
[1] 4 4 4 4 4 4 4 4 4 4  
rep(0:9,4)  
[1] 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4  
[36] 5 6 7 8 9

* If the second argument is a vector whose length is the same as the length of the first argument, then the elements of the two vectors are matched up 1-to-1. If each element of the vector in the second argument is the same, then the elements of the first argument are all repeated the same number of times, but individually this time, not as a unit.

rep(0:9,rep(4,10))  
[1] 0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5 6 6 6 6 7 7 7 7 8 8 8  
[36] 8 9 9 9 9

In specifying the repetition pattern for the second argument I used a shortcut. I wrote rep(4,10) which produces, as shown above, a vector of all 4s that is of length 10.

Now we're ready to recreate the data. To do this I make the second argument of rep the observed frequencies shown in the second column of Table 1.

num.stems<-c(6,8,9,6,6,2,5,3,1,4)

**#error check: numbers should sum to 50**

sum(num.stems)

[1] 50

aphid.data<-rep(0:9,num.stems)

aphid.data

[1] 0 0 0 0 0 0 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 4 4 4 4 4 4  
[36] 5 5 6 6 6 6 6 7 7 7 8 9 9 9 9

Given that there are no gaps in the count categories, we can visualize the distribution of the tabulated data with a bar chart using the barplot function.

barplot(num.stems)

To get labels on the categories we need to add a names attribute to the vector num.stems.

names(num.stems)<-0:9

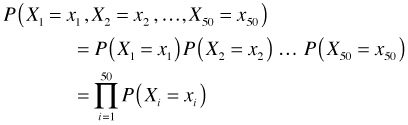
barplot(num.stems)

|  |  |
| --- | --- |
| (a) fig 1a | (b) fig 1b |
| **Fig. 1** Bar plots of the tabulated data: (a) without and (b) with the names attribute of the vector | |

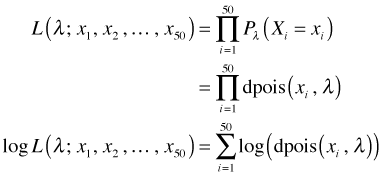
## Poisson model

### Constructing the Poisson log-likelihood

We have a random sample of 50 observations—the counts of the number of aphids on 50 different plant stems. Let Xi = # of aphids on stem i and let the observed values in our sample be denoted x1, x2, … , x50. The joint probability of obtaining the sample we got can be written generically as follows.



Our first goal today is to fit a Poisson distribution to these data. Once we replace the generic probability expression in the above expression above with a hypothesized probability model, we refer to the new expression as a likelihood. When we take the log of the likelihood we obtain the log-likelihood.



The quantity inside the parentheses is just the dpois function evaluated at xi and λ. The dpois function is listable. Thus for a fixed value of λ we can obtain the probabilities for all of our observations at once. For example, when λ = 1 we have

dpois(aphid.data,lambda=1)

[1] 3.678794e-01 3.678794e-01 3.678794e-01 3.678794e-01 3.678794e-01  
[6] 3.678794e-01 3.678794e-01 3.678794e-01 3.678794e-01 3.678794e-01  
[11] 3.678794e-01 3.678794e-01 3.678794e-01 3.678794e-01 1.839397e-01  
[16] 1.839397e-01 1.839397e-01 1.839397e-01 1.839397e-01 1.839397e-01  
[21] 1.839397e-01 1.839397e-01 1.839397e-01 6.131324e-02 6.131324e-02  
[26] 6.131324e-02 6.131324e-02 6.131324e-02 6.131324e-02 1.532831e-02  
[31] 1.532831e-02 1.532831e-02 1.532831e-02 1.532831e-02 1.532831e-02  
[36] 3.065662e-03 3.065662e-03 5.109437e-04 5.109437e-04 5.109437e-04  
[41] 5.109437e-04 5.109437e-04 7.299195e-05 7.299195e-05 7.299195e-05  
[46] 9.123994e-06 1.013777e-06 1.013777e-06 1.013777e-06 1.013777e-06

Next we need the log of all these values.

log(dpois(aphid.data,lambda=1))

[1] -1.000000 -1.000000 -1.000000 -1.000000 -1.000000 -1.000000  
[7] -1.000000 -1.000000 -1.000000 -1.000000 -1.000000 -1.000000  
[13] -1.000000 -1.000000 -1.693147 -1.693147 -1.693147 -1.693147  
[19] -1.693147 -1.693147 -1.693147 -1.693147 -1.693147 -2.791759  
[25] -2.791759 -2.791759 -2.791759 -2.791759 -2.791759 -4.178054  
[31] -4.178054 -4.178054 -4.178054 -4.178054 -4.178054 -5.787492  
[37] -5.787492 -7.579251 -7.579251 -7.579251 -7.579251 -7.579251  
[43] -9.525161 -9.525161 -9.525161 -11.604603 -13.801827 -13.801827  
[49] -13.801827 -13.801827

Finally we need to sum up all these values.

sum(log(dpois(aphid.data,lambda=1)))

[1] -215.9158

The last line is what we need for our log-likelihood function. We just need to turn it into a function in which lambda is the sole argument.

poisson.LL<-function(lambda) sum(log(dpois(aphid.data,lambda)))

poisson.LL(1)

[1] -215.9158

poisson.LL(2)

[1] -146.0014

The function seems to work.

### Graphing the log-likelihood

To graph the log-likelihood, we need to evaluate our function on a whole list of possible values of λ and then plot the results against λ. For example to plot over the range 1 ≤ λ ≤ 6, we could first generate values for λ in increments of 0.1 with seq(1,6,.1). But when I apply the log-likelihood function to it I can see something went wrong.

poisson.LL(seq(1,6,.1))

[1] -89.3177

R returns a single number. I was expecting it to return a list of numbers one for each value of seq(1,6,.1). What if I give it two numbers?

poisson.LL(1:2)

[1] -179.2257

poisson.LL(1)

[1] -215.9158

poisson.LL(2)

[1] -146.0014

As we can see when I give it both 1 and 2 the answer I get is neither the value at 1 nor the value at 2. To understand what's happening let's tease apart the function into its components. I repeat the above exercise using just the innermost part of our function, dpois.

dpois(aphid.data,1)

[1] 3.678794e-01 3.678794e-01 3.678794e-01 3.678794e-01 3.678794e-01  
[6] 3.678794e-01 3.678794e-01 3.678794e-01 3.678794e-01 3.678794e-01  
[11] 3.678794e-01 3.678794e-01 3.678794e-01 3.678794e-01 1.839397e-01  
[16] 1.839397e-01 1.839397e-01 1.839397e-01 1.839397e-01 1.839397e-01  
[21] 1.839397e-01 1.839397e-01 1.839397e-01 6.131324e-02 6.131324e-02  
[26] 6.131324e-02 6.131324e-02 6.131324e-02 6.131324e-02 1.532831e-02  
[31] 1.532831e-02 1.532831e-02 1.532831e-02 1.532831e-02 1.532831e-02  
[36] 3.065662e-03 3.065662e-03 5.109437e-04 5.109437e-04 5.109437e-04  
[41] 5.109437e-04 5.109437e-04 7.299195e-05 7.299195e-05 7.299195e-05  
[46] 9.123994e-06 1.013777e-06 1.013777e-06 1.013777e-06 1.013777e-06

dpois(aphid.data,2)

[1] 0.1353352832 0.1353352832 0.1353352832 0.1353352832 0.1353352832  
[6] 0.1353352832 0.2706705665 0.2706705665 0.2706705665 0.2706705665  
[11] 0.2706705665 0.2706705665 0.2706705665 0.2706705665 0.2706705665  
[16] 0.2706705665 0.2706705665 0.2706705665 0.2706705665 0.2706705665  
[21] 0.2706705665 0.2706705665 0.2706705665 0.1804470443 0.1804470443  
[26] 0.1804470443 0.1804470443 0.1804470443 0.1804470443 0.0902235222  
[31] 0.0902235222 0.0902235222 0.0902235222 0.0902235222 0.0902235222  
[36] 0.0360894089 0.0360894089 0.0120298030 0.0120298030 0.0120298030  
[41] 0.0120298030 0.0120298030 0.0034370866 0.0034370866 0.0034370866  
[46] 0.0008592716 0.0001909493 0.0001909493 0.0001909493 0.0001909493

dpois(aphid.data,1:2)

[1] 3.678794e-01 1.353353e-01 3.678794e-01 1.353353e-01 3.678794e-01  
[6] 1.353353e-01 3.678794e-01 2.706706e-01 3.678794e-01 2.706706e-01  
[11] 3.678794e-01 2.706706e-01 3.678794e-01 2.706706e-01 1.839397e-01  
[16] 2.706706e-01 1.839397e-01 2.706706e-01 1.839397e-01 2.706706e-01  
[21] 1.839397e-01 2.706706e-01 1.839397e-01 1.804470e-01 6.131324e-02  
[26] 1.804470e-01 6.131324e-02 1.804470e-01 6.131324e-02 9.022352e-02  
[31] 1.532831e-02 9.022352e-02 1.532831e-02 9.022352e-02 1.532831e-02  
[36] 3.608941e-02 3.065662e-03 1.202980e-02 5.109437e-04 1.202980e-02  
[41] 5.109437e-04 1.202980e-02 7.299195e-05 3.437087e-03 7.299195e-05  
[46] 8.592716e-04 1.013777e-06 1.909493e-04 1.013777e-06 1.909493e-04

If you compare the output from the three function calls. you'll see that in the last call R alternates its use of λ = 1 and λ = 2. For the first observation it uses 1, for the second it uses 2, for the third it uses 1, etc. Thus R is trying to match up the entries in each vector argument and when it runs out of entries in the second vector it just recycles the vector over and over again. Of course when we take the log of the results and then sum everything up we get total gibberish.

The solution is to force R to use each element of the λ vector one at a time. The sapply function will do this.

poisson.LL(1)

[1] -215.9158

poisson.LL(2)

[1] -146.0014

sapply(1:2,poisson.LL)

[1] -215.9158 -146.0014

Now we get the right answer.

So now we're ready to do the plot. To obtain the y-coordinates of the graph we need to use sapply. I specify type='l' in the plot function so that the plotted points are connected by line segments thus generating a curve. I use the xlab and ylab arguments to add labels to the axes.

plot(seq(2,5,.01), sapply(seq(2,5,.01), poisson.LL), type='l', xlab=expression(lambda), ylab='log-likelihood')

From the graph it would appear that the log-likelihood takes on its maximum value somewhere in the interval 3 ≤ λ ≤ 4. We obtain the same information if we plot the likelihood instead. The prod function replaces the sum function in the function definition of the likelihood.

poisson.L<-function(lambda) prod(dpois(aphid.data,lambda))

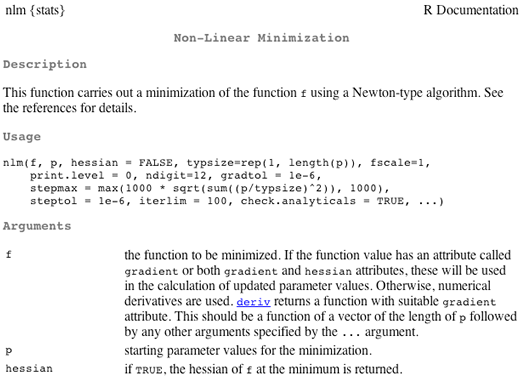
plot(seq(2,5,.01), sapply(seq(2,5,.01), poisson.L), type='l', xlab=expression(lambda), ylab='likelihood')

|  |  |
| --- | --- |
| (a) fig 2a | (b) fig 2b |
| **Fig. 2** Plots of the (a) log-likelihood and (b) likelihood functions for a Poisson model for the aphid count data | |

### Maximizing the log-likelihood numerically

We could zoom in on the MLE graphically and obtain an estimate that is as accurate as we please. It can also be shown using calculus that the MLE for λ in a Poisson distribution is the sample mean http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/xbar.gif. Even without this information we can obtain accurate estimates of the MLE using a numerical optimization routine. R provides two numerical optimization functions, nlm and optim, of which I focus on nlm. Figure 3 shows the help screen for nlm.

?nlm



**Fig. 3** Help screen for nlm

From the help screen we see that there are only two arguments that are required, f, the function to be minimized, and p, a vector of initial estimates of the parameter values. The rest of the arguments have default values.

Observe that nlm carries out minimization, not maximization. Thus in order for us to be able to use nlm we need to change the way we're formulating our problem. Since maximizing f is equivalent to minimizing –f, we need to reformulate our objective function so that it returns the negative log-likelihood rather than the log-likelihood.

poisson.negloglik<-function(lambda) -poisson.LL(lambda)

The nlm function uses a method similar to the Newton-Raphson method described in [lecture 6](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture6.htm#Newton). To use nlm we enter the name of our function, poisson.negloglik, as the first argument followed by an initial estimate for its single parameter lambda. Since our graphical analysis indicated that the MLE is near 3, I choose 3 for the second argument.

out.pois <- nlm(poisson.negloglik,3)

out.pois

$minimum  
[1] 124.1764

$estimate  
[1] 3.459998

$gradient  
[1] 6.571497e-08

$code  
[1] 1

$iterations  
[1] 4

The output tells us the following.

1. The value of the negative log-likelihood at the minimum is 124.1764.
2. The estimate of the MLE of λ is 3.459998.
3. The value of the gradient at the MLE is very small. This is what we want. The gradient is the same as the score, the derivative of the log-likelihood, which from calculus should be zero at a maximum or a minimum. Thus the fact that the value of the gradient is very close to zero encourages us to believe that nlm has found a solution (although there's no guarantee that it's the global minimum that we seek).
4. The help screen tell us that code=1 means "relative gradient is close to zero, current iterate is probably solution."
5. It took 4 iterations of the numerical algorithm to converge.

Using calculus the exact MLE for λ in a Poisson distribution is the sample mean http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/xbar.gif, which turns out to be the same as the estimate returned by nlm.

mean(aphid.data)

[1] 3.46

### ****Assessing the fit graphically****

Having estimated the Poisson model we next examine how well the model fits the observed data. A good place to start is by calculating the count frequencies predicted by the model and comparing these to the observed frequencies graphically. This can be accomplished with a bar plot of the observed frequencies on which we superimpose the Poisson predictions on top of the bars as points connected by line segments.

I begin by calculating the expected values under a Poisson model using the estimated value of λ. I generate the Poisson probabilities for category counts 0 to 9 (matching the range of observed values we have). To obtain the expected counts I multiply these numbers by 50, the number of observations. (Note: What I'm really doing is adding up the predicted probability distributions for all 50 observations. Because each observation has the same predicted probability distribution, this is the same as multiplying that single probability distribution by 50.)

exp<-dpois(0:9,out.pois$estimate)\*50

max(exp)

[1] 10.84896

max(table(aphid.data))

[1] 9

Notice that the expected values have a larger maximum than the observed values. To ensure that the expected values (which will be added to the graph with the points function) are displayed completely I will need to add the ylim argument to barplot to extend the y-axis to include the maximum expected count.

If we sum the expected counts we find that they don't add up to 50. Equivalently, if we sum the predicted probability distribution, the probabilities don't sum to one.

sum(dpois(0:9,out.pois$estimate))

[1] 0.9969393

That's because we're leaving off the tail probability. The Poisson distribution is unbounded above, so there is a nonzero probability of obtaining an observation with 10 or more counts. We can obtain this probability with the ppois function.

**#P(X>9) = 1 - P(X <= 9)**

1-ppois(9,out.pois$estimate)

[1] 0.003060710

So, we have a choice. We can create an additional category for both the observed and predicted counts, labeling it '10+' perhaps, or we can just lump P(X>9) into the last category effectively making it represent '9 or more'.

**#choice 1: modify observed and expected so that there are 11 categories**

obs.counts<-c(num.stems,0)

exp.counts<-c((dpois(0:9,out.pois$estimate)),1-ppois(9,out.pois$estimate))\*50

rbind(obs.counts, exp.counts)

0 1 2 3 4 5 6 7  
obs.counts 6.000000 8.000000 9.00000 6.00000 6.000000 2.000000 5.000000 3.000000  
exp.counts 1.571491 5.437355 9.40662 10.84896 9.384349 6.493966 3.744852 1.851026  
8 9   
obs.counts 1.0000000 4.0000000 0.0000000  
exp.counts 0.8005683 0.3077739 0.1530355

**#choice 2: add NB tail to last observed category**

exp.pois<-c((dpois(0:8,out.pois$estimate)), 1-ppois(8,out.pois$estimate))\*50

rbind(num.stems, exp.pois)

0 1 2 3 4 5 6 7  
num.stems 6.000000 8.000000 9.00000 6.00000 6.000000 2.000000 5.000000 3.000000  
exp.pois 1.571491 5.437355 9.40662 10.84896 9.384349 6.493966 3.744852 1.851026  
8 9  
num.stems 1.0000000 4.0000000  
exp.pois 0.8005683 0.4608094

Notice that in the second choice I calculate dpois(0:8) and then use 1-ppois(8) for the last observed category. Either of these choices is reasonable. In what follows I use the second option. I verify that the expected counts now sum to 50.

sum(exp.pois)

[1] 50

In order to add the expected counts to the bar chart I need to know the x-coordinates of the centers of the bars. You might think these are just the labels that appear below the bars, but in fact the labels shown need not correspond to the coordinate system R has used. Fortunately the coordinates used are returned by R when you assign the results of the barplot function to an object. The coordinates of the bars are the return value of the barplot function.

out.bar<-barplot(num.stems, ylim=c(0,11))

out.bar

[,1]  
[1,] 0.7  
[2,] 1.9  
[3,] 3.1  
[4,] 4.3  
[5,] 5.5  
[6,] 6.7  
[7,] 7.9  
[8,] 9.1  
[9,] 10.3  
[10,] 11.5

I use the barplot coordinates as the x argument of the points function. The type='o' option specifies that point symbols should be overlaid on top of line segments connecting the points.

points(out.bar, exp.pois, pch=16, cex=.9, type='o')

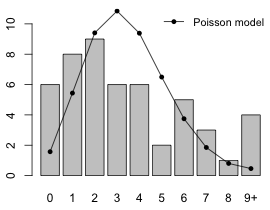
We probably should relabel the last category to indicate that it is '9 or more' rather than just 9. The barplot function has an argument names.arg that can be used to modify the labels appearing under the bars. I also add a legend that identifies the model being fit.

**#relabel bars**

out.bar<-barplot(num.stems, ylim=c(0,11), names.arg=c(0:8,'9+'))

points(out.bar, exp.pois, pch=16, cex=.9, type='o')

legend('topright', 'Poisson model', pch=16, col=1, lty=1, cex=.9, bty='n')



**Fig. 4** Fit of the Poisson model

Based on the plot the fit doesn't look very good.

## Negative binomial model

### Constructing the negative binomial log-likelihood

Following an argument that is completely analogous to the one presented for the Poisson distribution, the log-likelihood for a negative binomial distribution applied to these data is the following.

dnbinom

Translating this into an R function yields the following.

**#negative binomial log-likelihood**

NB.LL<-function(mu,theta) sum(log(dnbinom(aphid.data, mu=mu, size=theta)))

As you can see the only difference from the Poisson function is that we need to specify two arguments, the mean and the dispersion parameter. Instead of specifying the two arguments as separate variables, we can specify them as components of a vector.

**#alternative vector version**

NBvec.pos<-function(p) sum(log(dnbinom(aphid.data, mu=p[1], size=p[2])))

The next few lines contrast how these two versions of the negative binomial log-likelihood function differ.

**#first function**

NB.LL(3,4)

[1] -116.1379

**#incorrect use of 2nd function with two separate arguments**

NBvec.pos(3,4)

Error in NBvec.pos(3, 4) : unused argument(s) ( ...)

**#correct usage in which arguments are entered as a vector**

NBvec.pos(c(3,4))

[1] -116.1379

The nlm function does minimization so our function needs to return the negative of the log-likelihood. The nlm function also requires that the function have a single vector argument of parameters, i.e., the second version of the log-likelihood function given above. I use the function NB.LL created above to define a negative log-likelihood negative binomial function.

**#negative log-likelihood for nlm**

negNB.LL<-function(p) -NB.LL(p[1],p[2])

Alternatively we can write this last function out using multiple lines in a way that will make it easy to generalize things to a regression setting.

**# writing the same function expanded over multiple lines**

negNB.LL<-function(p){

mu<-p[1]

theta<-p[2]

LL<-sum(log(dnbinom(aphid.data, mu=mu, size=theta)))

-LL

}

### Obtaining the MLE

I use the nlm function and the negative log-likelihood function created above to numerically approximate the MLEs. I supply initial guesses for http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/muhat.gifand http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/thetahat.gif.

out.NB <- nlm(negNB.LL, c(3,4), hessian=TRUE)

out.NB

$minimum  
[1] 114.7009

$estimate  
[1] 3.459995 2.645024

$gradient  
[1] -2.251151e-05 1.317917e-05

$hessian  
[,1] [,2]  
[1,] 6.2589597977 0.0002248452  
[2,] 0.0002248452 0.9535296142

$code  
[1] 1

$iterations  
[1] 8

From the output we see the algorithm converged (code = 1) and both components of the gradient are close to zero.

There is a second optimization function available in R called optim. This function has many more options than does nlm. Its syntax is similar except that the order of the two arguments we've been using is reversed. Here's what we get using optim.

optim(c(3,4), negNB.LL)

$par  
[1] 3.460087 2.645439

$value  
[1] 114.7009

$counts  
function gradient   
49 NA

$convergence  
[1] 0

$message  
NULL

The output is organized differently, but the results match those returned by nlm.

### Assessing the fit of the negative binomial model graphically

I produce a graph that compares the fit of the negative binomial and Poisson models. First I examine the predicted probabilities and counts.

dnbinom(0:9, mu=out.NB$estimate[1], size=out.NB$estimate[2])

[1] 0.10943985 0.16405655 0.16945422 0.14869883 0.11893284 0.08958118   
[7] 0.06468935 0.04527822 0.03093792 0.02073880

dnbinom(0:9, mu=out.NB$estimate[1], size=out.NB$estimate[2])\*50

[1] 5.471993 8.202827 8.472711 7.434941 5.946642 4.479059 3.234468 2.263911   
[9] 1.546896 1.036940

**#anything left over?**

sum(dnbinom(0:9,mu=out.NB$estimate[1], size=out.NB$estimate[2])\*50)

[1] 48.09039

I combine the counts 9 and above into the last category

NB.p <-c (dnbinom(0:8, mu=out.NB$estimate[1], size=out.NB$estimate[2]), 1-pnbinom(8, mu=out.NB$estimate[1], size=out.NB$estimate[2]))

NB.p\*50->exp.NB

I rename the last category so that it correctly indicates 9 and above.

names(num.stems)[length(num.stems)]<-"9+"

I next produce a bar graph that compares the fit of the Poisson and negative binomial models.

out.bar<-barplot(num.stems, ylim=c(0, max(c(exp.NB, exp.pois, num.stems))))

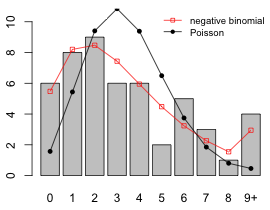
**#add negative binomial**

points(out.bar, exp.NB, col=2, pch=22, cex=.9, type='o')

**#add poisson**

points(out.bar, exp.pois, col=1, pch=16, cex=.9, type='o')

legend('topright', c('negative binomial', 'Poisson'), col=c(2,1), lty=1, pch=c(22,16), bty='n', cex=.8)

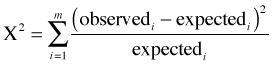


**Fig. 5** Fit of the Poisson and negative binomial models

Based on the plot the fit of the negative binomial model appears to be a big improvement over that of the Poisson.

## Pearson chi-squared goodness of fit test for discrete data

Graphically the fit of the negative binomial looks far better than that of the Poisson. We can test this formally with the Pearson chi-squared goodness of fit test. The Pearson chi-squared test compares the observed category frequencies to those predicted by a model (referred to as the expected frequencies) using the following formula.



where m is the number of categories. It turns out the Pearson chi-squared statistic has a chi-squared distribution.

http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/pearsondistribution.gif

The degrees of freedom are the number of categories, m, minus one minus the number of estimated parameters, p, used in obtaining the expected frequencies.

The null hypothesis of this test is that the fit is adequate.

H0: model fits the data  
H1: model does not fit the data

We should reject the null hypothesis at level α if the observed value of our test statistic exceeds the 1 – α quantile of a chi-squared distribution with m – 1 – p degrees of freedom.

Reject if http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/rejectnull.gif

The chi-squared distribution of http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/X2.gifis an asymptotic result. For it to hold the fitted values, the expected frequencies obtained using the model, should be large. A general rule is that expected cell counts should be 5 or larger, although with many categories Agresti (2002) notes that an expected cell frequency as small as 1 is okay as long as no more than 20% of the expected counts are less than 5. The difficulty in applying this rule when fitting a model such as the Poisson model to data is that the theoretical count distribution is infinite (although the probabilities after a certain point are essentially zero). So the 20% guideline is not well-defined.

For our data quite a few of the expected counts are small. In the code below I calculate the expected frequencies, http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/expectedcounts.gif, for Poisson counts ranging from 0 to 12. From the output we see that the expected frequencies exceed 5 only for k = 1, 2, 3, 4, and 5.

dpois(0:12,out.pois$estimate)\*50

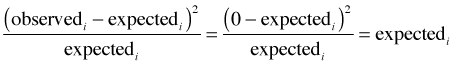
[1] 1.571490812 5.437355498 9.406620322 10.848963362 9.384348629  
[6] 6.493966014 3.744851868 1.851025857 0.800568284 0.307773876  
[11] 0.106489708 0.033495837 0.009657961

What to do in such a situation is not entirely clear. The standard recommendation, cited for example in Sokal and Rohlf (1995) is to pool categories. They write, p. 702, "Whenever classes with expected frequencies of less than five occur, expected and observed frequencies for those classes are generally pooled with an adjacent class to obtain a joint class with an expected frequency http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/fhat.gif." The number of categories m is thereby reduced in the degrees of freedom of the chi-squared distribution. Not everyone agrees with this. Here's a sampling from the literature.

1. These authors recommend pooling but don't give an absolute criterion: "…it is insightful to compare predicted relative frequencies http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/phati.gifwith actual relative frequencies http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/pbari.gif. These are given in Table 5.6 where counts of five or more are grouped into the one cell to prevent cell sizes from getting too small." Cameron & Trivedi (1998), p. 157.
2. This author uses a frequency of 1 as the cutoff: "It is also recommended that adjacent groups at the bottom of the table be combined in order to avoid having any expected frequencies less than 1." Le (1998), p. 209.
3. This author says never pool: "We note here that several of the fitted values are small (say, less than 5), and it is a common practice to pool over categories, to obtain larger expected values. This is somewhat arbitrary and potentially dangerous, and should be avoided if possible." Morgan (2000), p. 17.

My comments on this are the following.

1. Pooling is unavoidable. Since tail probabilities for discrete count models run forever, it is always necessary to at least pool the observations in the tail. It is not clear though where this should begin without additional guidelines, but a sensible approach would be to add the expected tail probabilities to the last category for which the observed counts are also nonzero. My rationale for this is the following. Every additional cell beyond the last for which the observed count is zero will add the value of the expected count to the chi-squared statistic, no matter how small the expected count, while at the same time add 1 to the degrees of freedom. To see this consider the generic term of the Pearson chi-squared test when the observed count is 0.



If there are many such cases the effective result will be to dilute a significant lack of fit that may be occurring if we had just looked at the nonzero cells. (Generally speaking, a chi-squared random variable does not become significant until it exceeds twice its degrees of freedom. Each of these terms adds one degree of freedom to the test statistic. If at the same time the expected frequency is less than 2 then these terms will reduce rather than increase the lack of fit as measured by the test statistic.) So I think pooling these cells in the tail makes sense.

1. There is no doubt that when the expected cell counts are small the asymptotic chi-squared distribution is a bad approximation to the true distribution of the test statistic.
2. The reason you carry out the Pearson test is to provide evidence that your model fits the data. If you fail to find a significant lack of fit and you chose to pool (or not) then you need to investigate whether pooling had an effect on the result you obtained. This is the fundamental difference between model falsification and model confirmation. We can falsify models conclusively, but we can't unequivocally confirm them. If the manner in which the pooling was done was arbitrary, then pooling in some other way is a good check on your results. The bottom line is that you need to convince a skeptic that you tried your best to falsify your model but were unable to do so.
3. Randomization tests may be an option when there are small expected cell counts. I discuss this approach [below](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#poissonsimulate)

## Testing the fit of the Poisson model

### ****Parametric goodness of fit test****

Using 5 as a cut-off we see that we should combine the first two expected counts corresponding to k = 0 and k = 1. Turning to the right hand tail of the distribution, k = 6 is the first expected count to drop under 5. If we calculate the sum of the expected counts greater than 6 we see that these don't exceed 5 either. So we should combine all category counts for k ≥ 6

**# count X = 6**

dpois(6,out.pois$estimate)\*50

[1] 3.744852

**# counts X > 6**

(1-ppois(6,out.pois$estimate))\*50

[1] 3.112403

Note: ppois(6,out.pois$estimate)) = P(X ≤ 6). Therefore, 1–ppois(6,out.pois$estimate)) = P(X > 6). So, I add X = 6 to the tail too.

expected.p <- c(sum(dpois(0:1,out.pois$estimate)), dpois(2:5,out.pois$estimate), 1-ppois(5,out.pois$estimate))

expected <- expected.p\*50

expected

[1] 7.008846 9.406620 10.848963 9.384349 6.493966 6.857255

I next group the observed counts in exactly the same way: pool the first two, keep the next four separate, and pool the remainder. I use length(num.stems) to locate the position of the last value.

observed<-c(sum(num.stems[1:2]), num.stems[3:6], sum(num.stems[7:length(num.stems)]))

observed

2 3 4 5   
14 9 6 6 2 13

Finally I carry out the Pearson chi-squared test. It can be calculated by hand or extracted from the output of the chisq.test function. The hand calculations are as follows.

pearson<-sum((observed-expected)^2/expected)

pearson

[1] 18.99147

df<-length(observed)-1-1

df

[1] 4

p.val<-1-pchisq(pearson,df)

p.val

[1] 0.0007889844

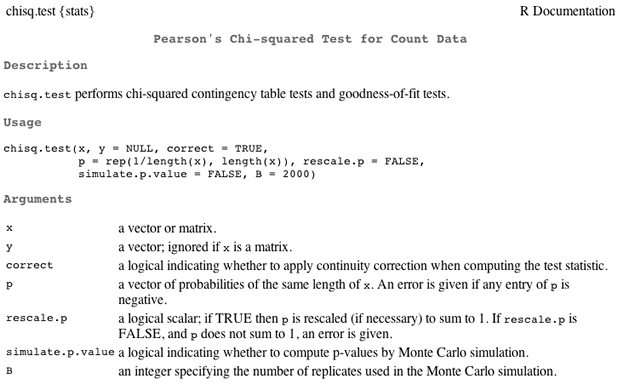
qchisq(.95,df)

[1] 9.487729

The p-value is quite small indicating a significant lack of fit. The critical value for our test is 9.49 and the observed value of our test statistic is 18.99. As a general rule for a chi-squared test, if the test statistic is more than twice its degrees of freedom, the p-value will end up being significant.

We can also use the chisq.test to perform the Pearson chi-square goodness of fit test. Its help screen is shown below (Fig. 6).

?chisq.test



**Fig. 6** Help screen for chisq.test

The entries relevant to us are x and p. The x= argument should contain the observed counts. The p= argument should contain the expected probabilities (not the expected counts). I try it for our grouped data.

chisq.test(observed, p=expected.p)

Chi-squared test for given probabilities

data: observed   
X-squared = 18.9915, df = 5, p-value = 0.001929

Observe that while the value of the test statistic is correct, the p-value is wrong as are the degrees of freedom. That's because the function doesn't know we estimated a parameter in order to obtain the expected probabilities. We can correct this by using chisq.test to obtain the Pearson statistic, but then calculate the p-value ourselves. I rerun the chi-squared test and save the result as an object in R.

out.chisq <- chisq.test(observed, p=expected.p)

names(out.chisq)

[1] "statistic" "parameter" "p.value" "method" "data.name" "observed"   
[7] "expected" "residuals"

out.chisq$statistic

X-squared   
18.99147

out.chisq$parameter

df   
5

The $statistic component contains the Pearson statistic. The $parameter component contains the number of categories minus one, m – 1. Because we estimated one parameter the degrees of freedom should be decreased by one more.

1-pchisq(out.chisq$statistic, df=out.chisq$parameter-1)

X-squared   
0.0007889844

This matches the hand-calculation carried out above. So, we reject the null hypothesis and conclude that there is a significant lack of fit.

### ****Simulation-based goodness of fit test****

The chisq.test has an additional argument called simulate.p.value. Setting this argument to TRUE causes R to carry out a Monte Carlo simulation to obtain the p-value of the observed value of the test statistic http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/X2.gif. This works as follows.

1. The expected probability vector **p** is used to randomly generate new data. In the current problem **p** has ten components. Let the ten components of **p** be labeled p0, p1, ... , p9. At each step of the simulation we generate a new observation. The new observation has probability p0 of being a 0, probability p1 of being a 1, etc. Thus we are essentially generating a random observation from a specified multinomial distribution. This step can be easily programmed by using a uniform random number generator.
2. Repeat this as many times as there are observations. That means with the current data set the procedure should be carried out 50 times, each time obtaining one of the values 0 to 9. These 50 simulated observations become the new set of raw counts.
3. Next calculate http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/X2.gifusing these simulated data as if they were the observed values. The expected values are obtained, as usual, by multiplying the components of **p** by 50. Denote this chi-squared statistic http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/x2sim.gif.
4. Step 3 is carried out a total of B times (by default B = 2000). Each time a new set of 50 simulated observations is generated and the corresponding http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/X2.gifstatistic is calculated. So in the end there will be B = 2000 values of http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/x2sim.gif.
5. Calculate http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/X2.gifusing the actual observations. Denote this value by http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/x2obs.gifand add it the vector of http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/x2sim.gifvalues.
6. Finally count up the number of http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/x2sim.gifvalues that are greater than or equal to http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/x2obs.gif, (there will always be one such value because http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/x2obs.gifis included among the http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/x2sim.gifvalues in step 5), and divide the result by B + 1. This is the simulation-based p-value.

The beauty of this approach is that it avoids the whole issue of whether the asymptotic chi-squared distribution is appropriate or not, because it doesn't use it! The set of simulated values defines the distribution of the http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/X2.gifstatistic under the null hypothesis that the model fits the data. The issue of pooling doesn't arise, because it's not necessary. Having said that it is still necessary to combine the tail probabilities of the probability model in order to create the vector **p** used in the simulation.

I redo the goodness of fit test this time setting simulate.p.value=TRUE. I calculate the predicted probabilities using the Poisson model adding the tail probability to the last observed category, X = 9.

poisson.p <- c((dpois(0:8, out.pois$estimate)), 1-ppois(8, out.pois$estimate))

chisq.test(num.stems, p=poisson.p, simulate.p.value=TRUE)

Chi-squared test for given probabilities with simulated p-value  
(based on 2000 replicates)

data: num.stems   
X-squared = 48.5686, df = NA, p-value = 0.0004998

Observe that the reported p-value is equal to http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/pval1.gifmeaning that the observed value of the test statistic was larger than any of the simulated values.

1/2001

[1] 0.0004997501

With 2000 simulations our p-value is accurate to 2 or 3 decimal places. Thus the best we can say is that the true p-value is probably less than 0.001. To get a more accurate result we can increase the number of simulations by specifying our own value for the argument B. I try 9,999 simulations. I choose 9,999 so that when we include the observed value of our test statistic as one of the simulations we get a nice round number of 10,000.

chisq.test(num.stems, p=poisson.p, simulate.p.value=TRUE, B=9999)

Chi-squared test for given probabilities with simulated p-value  
(based on 9999 replicates)

data: num.stems   
X-squared = 48.5686, df = NA, p-value = 3e-04

The reported p-value is equal to http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/pval2.gifand tells us that there were two additional simulated values of http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/x2sim.gifas extreme as http://www.unc.edu/courses/2010fall/ecol/563/001/images/lectures/lecture7/x2obs.gif. We now have roughly 3-decimal place accuracy for our p-value and can report p < .001. In any case it is clear, both graphically and analytically, that the Poisson model does not provide an adequate fit to these data.

## Testing the fit of the negative binomial model

### Parametric goodness of fit test

If we examine the expected counts we see that there are a few below 5.

exp.NB

[1] 5.471993 8.202827 8.472711 7.434941 5.946642 4.479059 3.234468  
[8] 2.263911 1.546896 2.946552

As was noted above if too many of the expected counts are small (5 being a guideline for smallness) then the chi-squared distribution of the Pearson statistic becomes questionable. To deal with this I combine categories. One way to do this would be to leave the first five categories alone, combine the next two, and combine the last three. A good strategy is to start with the tail and work backwards.

NB.p.new<-c(NB.p[1:5],sum(NB.p[6:7]),sum(NB.p[8:10]))

NB.merge <- 50\*NB.p.new

[1] 5.471993 8.202827 8.472711 7.434941 5.946642 7.713526 6.757360

I merge the same categories of the observed counts and carry out the test. The degrees of freedom for the chi-squared statistic is m – 1 – p where m is the number of categories (after merging) and p is the number of estimated parameters. We have two estimated parameters: μ and θ so p = 2.

Oi.merge<-c(num.stems[1:5], sum(num.stems[6:7]), sum(num.stems[8:10]))

Pearson<-sum((Oi.merge-NB.merge)^2/NB.merge)

Pearson

[1] 0.6607193

Pearson.p<-1-pchisq(Pearson,length(Oi.merge)-1-2)

Pearson.p

[1] 0.9560832

We can also use the output from the chisq.test function to obtain the Pearson statistic.

chisq.NB <- chisq.test(Oi.merge, p=NB.p.new)

names(chisq.NB)

[1] "statistic" "parameter" "p.value" "method" "data.name" "observed"   
[7] "expected" "residuals"

1-pchisq(chisq.NB$statistic, df=chisq.NB$parameter-2)

X-squared   
0.9560832

The p-value is large so we fail to reject the fit of the negative binomial model.

### Simulation-based goodness of fit test

By pooling categories we have distorted the negative binomial distribution. Failing to find a significant lack of fit with the pooled categories does not guarantee that the model would still fit if we didn't pool the categories. We can obtain a better measure of fit by using a randomization test.

chisq.test(num.stems, p=NB.p, simulate.p.value=TRUE, B=9999)

Chi-squared test for given probabilities with simulated p-value  
(based on 9999 replicates)

data: num.stems   
X-squared = 3.5113, df = NA, p-value = 0.9459

The result is nearly identical to what we obtained with the parametric test using the combined categories.

A skeptic might argue that the way we're handling the last category is overly favorable to our model. The empirical distribution of the raw data does not decrease to zero; the last observed category has four observations. By lumping the tail of the negative binomial distribution into the last observed category we force the expected results to more closely resemble the observed data. To satisfy such a skeptic I create a new category, X = 10+, for both the observed and expected counts and redo the randomization test. This category represents stems with aphid counts of 10 or more. Of course, the observed frequency of this new category is 0.

new.numstems<-c(num.stems,0)

new.NB.p<-c(dnbinom(0:9, mu=out.NB$estimate[1], size=out.NB$estimate[2]), 1-pnbinom(9, mu=out.NB$estimate[1], size=out.NB$estimate[2]))

new.NB.p

[1] 0.10943985 0.16405655 0.16945422 0.14869883 0.11893284 0.08958118  
[7] 0.06468935 0.04527822 0.03093792 0.02073880 0.03819224

new.NB.p\*50

[1] 5.471993 8.202827 8.472711 7.434941 5.946642 4.479059 3.234468  
[8] 2.263911 1.546896 1.036940 1.909612

chisq.test(new.numstems, p=new.NB.p, simulate.p.value=TRUE, B=9999)

Chi-squared test for given probabilities with simulated p-value  
(based on 9999 replicates)

data: new.numstems   
X-squared = 13.5113, df = NA, p-value = 0.1923

So even after creating a separate final category in which the observed count is zero, we still fail to find a significant lack of fit (although the p-value of our test has been reduced substantially).

**Warning**. I should point out one deficiency of the simulation-based test. There is no penalty for overfitting. In the Pearson chi-squared test the degrees of freedom of the chi-squared distribution used for assessing the significance of the test statistic gets reduced by the number of parameters that were estimated. This has the effect of reducing the threshold for lack of fit. As a result, models that use more parameters are forced to meet a higher goodness of fit standard than are models using fewer parameters. This makes it harder for complicated models to pass the test. Thus the parametric version of the chi-squared test has a built-in protection against over-fitting. With the simulation-based version of the test there is no such protection.

## Cited references

* Agresti, Alan. 2002. Categorical Data Analysis. Wiley: New York.
* Cameron, A. Colin and Pravin K. Trivedi. 1998. Regression Analysis of Count Data. Cambridge University Press: New York.
* Krebs, Charles. 1999. *Ecological Methodology*. Addison-Wesley: Menlo Park, CA.
* Le, Chap T. 1998. Applied Categorical Data Analysis. Wiley: New York.
* Morgan, Byron J. T. 2000. Applied Stochastic Modelling. Arnold Press: London.
* Sokal, Robert R. and F. James Rohlf. 1995. Biometry. W. H. Freeman: New York.

## R Code

A compact collection of all the R code displayed in this document appears [here](http://www.unc.edu/courses/2010fall/ecol/563/001/notes/lecture7%20Rcode.html).

[Course Home Page](http://www.unc.edu/courses/2010fall/ecol/563/001/index.html)

|  |
| --- |
| Jack Weiss *Phone:* (919) 962-5930 *E-Mail:* jack\_weiss@unc.edu *Address:* Curriculum in Ecology, Box 3275, University of North Carolina, Chapel Hill, 27516 Copyright © 2010 Last Revised--September 27, 2010 URL: [http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm](http://www.unc.edu/courses/2010fall/ecol/563/001/docs/lectures/lecture7.htm#lecture5) |