Homework 9, Fitting a Simulation Model to Data

36-467/667

Due at 6 pm on Thursday, 5 November 2020

In this problem set, we continue working with the lynx data set, and with the `lv` model from the previous problem set. This is a version of the **Lotka-Volterra** model, named after the scientists[1] who invented it. This model explains cycles in the population of an animal species by looking at the interactions between a predator species (like the lynx) and its prey. (The main prey of the lynx is a kind of rabbit called the "snowshoe hare".)

Specifically, the model is mostly a set of differential equations[2], like so:

$$\begin{aligned}
\frac{dS}{dt} &= (\theta_1 R(t) - \theta_2) S(t) && (1)\\
\frac{dR}{dt} &= (\theta_3 - \theta_4 S(t)) R(t) && (2)\\
S(0) &= 1 && (3)\\
R(0) &= 1 && (4)\\
X(t) &\sim \text{Poisson}(\theta_5 S(t)) && (5)
\end{aligned}$$

Here, $S(t)$ is the number of predators at time t, $R(t)$ is the number of prey animals at time t, and $X(t)$ is the number of predators observed (caught) at time t. $S(t)$ and $R(t)$ are measured relative to the abundance of these animals at time $t = 0$, but $X(t)$ is supposed to be a simple count.

The meaning of the model parameters is as follows. If there were no predators ($S(t) = 0$), the prey species would grow exponential in population at rate θ_3. The more predators there are, the lower the growth rate of the prey (because of the $-\theta_4 S(t)$ term), until, past some point, predators actually cause the prey population to shrink. If there were no prey ($R(t) = 0$), the population of predators would shrink exponentially at rate $-\theta_2$. Adding more prey animals raises the growth rate of the predators, whose numbers will grow exponentially if $\theta_1 R(t) > \theta_2$. Finally, θ_5 is an observational or measurement parameter, which tells us how the relative abundance of predators $S(t)$ relates to the expected number of observed predators $X(t)$. (Notice that all of the parameters, to make sense, must be > 0.)

In this problem set, we'll use the methods we've developed so far to estimate this model. You will want to refer to the `lv.R` code from last time.

1. (5) Explain why the first observed value for X is a reasonable rough or initial estimate of θ_5.
2. A **fixed point** of a dynamical system, like the Lotka-Volterra equations, is a state (S, R) where the rates of change are 0, so $dS/dt = dR/dt = 0$.

[1]Lotka was trained as a physicist, became a pioneer of applying mathematics to biology, couldn't find a university job and ended up spending most of his career working for a life insuance company (where he did pioneering work on demography). Volterra was an eminent pure mathematician who looked in to the problem of population cycles as a favor to his son-in-law, who worked for the Italian equivalent of the fish-and-wildlife department. Interestingly, neither was trained as an ecologist (a field which was just emerging when they wrote), or even a biologist.

[2]If you look this up in a textbook (or Wikipedia...), you will see something where the initial values of $S(t)$ and $R(t)$ are arbitrary, rather than fixed at 1. The usual version can however always be transformed in to the one I give here, and doing so saves us from having to estimate two parameters. Also, making the observable follow a Poisson distribution isn't essential; some people use different distributions of noise, but the idea that the mean is proportional to $S(t)$ *is* essential.

a. (2) Show that $(0, 0)$ is a fixed point, regardless of the parameters.

b. (3) Show that $(S, R) = (\theta_3/\theta_4, \theta_2/\theta_1)$ is a fixed point.

c. (2) Show that there are no other fixed points.

3. The `lv.latent()` function in the code from last week runs the part of the model which calculats $S(t)$ and $R(t)$ (but not $X(t)$). Run it with parameters $\theta_1 = 1.6$, $\theta_2 = 2.5$, $\theta_3 = 0.5$ and $\theta_4 = 0.1$, and times evenly spaced from 0 to 60 in units of 1/12. Save the result in an object called `lv.out`.

   a. (5) Which of the mathematical variables $R(t)$, $S(t)$ or $X(t)$ corresponds to the numerical R variable `lynx` in `lv.out`? Which mathematical variable corresponds to `hare` in `lv.out`?

   b. (5) Plot the `lynx` variable against time; it should look cyclical. How much time passes between peaks? What's the ratio between the minimum and maximum population of lynxes?

   c. (5) Plot the `lynx` variable against the `hare` variable. You should see a lop. Does the process move around the loop with equal speed in all phases of its cycle, or does it move faster in some parts and slower in others? How can you tell?

   d. (5) Add the fixed point corresponding to these parameter values to the plot. Describe where the point lies in relation to the trajectory.

Lotka and Volterra showed that time averages correspond to the fixed point. That is, that $T^{-1} \int_{t=0}^{T} S(t)dt \rightarrow \theta_3/\theta_4$ as $T \rightarrow \infty$, while $T^{-1} \int_{t=0}^{T} R(t)dt \rightarrow \theta_2/\theta_1$. That is, the system "orbits around" the fixed point, and the average as it does so matches the fixed point. (Not all dynamical systems which have fixed points have this property.)

4. (5) Explain why the time average of $X(t)$ is a reasonable rough, initial estimate of $\theta_5(\theta_3/\theta_4)$. What is the corresponding estimate of θ_3/θ_4 for the lynx data?

5. We have tried fitting linear autoregressive models to this data before, and found the results aren't very satisfying. One reason for this, hinted at in the last homework, is that there seems to be a lot more variability in the population count at the peaks of the cycle than at the bottom, and in fitting AR models we assume constant variance. We are nonetheless going to try using an autoregressive model as our auxiliary model for fitting the Lotka-Volterra equations to this data using indirect inference.

```
estimate.auxiliary <- function(x) {
  ar.fit <- ar.ols(x, order.max=3, aic=FALSE, demean=FALSE, intercept=TRUE)
  beta <- c(log(ar.fit$x.intercept), ar.fit$ar, log(sd(ar.fit$resid, na.rm=TRUE)))
  return(beta)
}
```

   a. (5) Explain what the `estimate.auxiliary` function does: what is its argument, what does it return, and what purpose does it serve here?

   b. (5) Some of the estimated auxiliary parameters are logged by this function, but others are not. Why do you think that's done? That is, why log those parameters but not the others?

   c. (5) Why does the `estimate.auxiliary` function use an AR(3) model, when there are 5 parameters in the Lotka-Volterra model? Could we use an AR(2) instead? What about an AR(4)?

   d. (5) Do any of the entries in β, as returned by this function, correspond to coordinates of the θ vector of parameters in the Lotka-Volterra model? Explain.

6. To estimate the model, we start with a guess about the Lotka-Volterra parameters, find the corresponding value of the auxiliary parameters by simulating, and then adjust the Lotka-Volterra parameters to match the data better. Ordinarily, we'd just use `optim()` to do the iterative adjustment. However, all the components of θ need to be > 0, so we need to use a method for optimization which can handle

inequality constraints. The `constrOptim()` function, in R, will do this[3]. The `lv.indinf()` function, given below and on the class website, uses this.

a. (3) The `lv.indinf()` function defines an internal function called `aux.discrep()`. What does it do? How is it used by the code?
b. (5) Run this code using `theta.ref` as the starting value of the parameters. What are the fitted values?
c. (6) Make *one* simulation run of the model at the fitted values. Plot this simulation run as a function of time, and add the original data to the plot. Comment on how the simulation run resembles the data and on how they differ.

```
lv.indinf <- function(start, data=lynx, s=100) {
    require(numDeriv)
    beta.hat <- estimate.auxiliary(data)
    aux.discrep <- function(theta) {
        aux.from.sims <- replicate(s, estimate.auxiliary(lv.sim(theta.vec=theta)))
        beta.bar <- rowMeans(aux.from.sims)
        sum((beta.hat - beta.bar)^2)
    }
    gradient.discrep <- function(theta) {
        return(grad(func=aux.discrep, x=theta, method="simple"))
    }
   est <- constrOptim(theta=start, f=aux.discrep, grad=gradient.discrep,
                      ui=diag(length(start)), ci=rep(0, length(start)),
                      method="BFGS")
    return(est)
}
```

7. Uncertainty.
   a. (5) Write code which runs the model at the fitted parameters from the previous problem, and then runs the indirect inference function *on the simulation.*
   b. (6) Write code which repeats your code from part (a) multiple times and takes the standard deviation across the simulation runs. **Hint**: For debugging purposes here, make the number of replications small.
   c. (6) We've talked about multiple ways of calculating standard errors for estimators in this course. What kind of standard error is this giving us?
   d. (2) Run your code from (b) to get standard errors for each of the five parameters of the Lotka-Volterra model. **NOTE**: This is the most time-consuming part of the assignment. Use caching, and observe how points have been assigned to problems.

Presentation rubric (10): The text is laid out cleanly, with clear divisions between problems and sub-problems. The writing itself is well-organized, free of grammatical and other mechanical errors, and easy to follow. All plots, tables, etc., are generated automatically by code embedded in the R Markdown file. Plots are carefully labeled, with informative and legible titles, axis labels, and (if called for) sub-titles and legends; they are placed near the text of the corresponding problem. All quantitative and mathematical claims are supported by appropriate derivations, included in the text, or calculations in code. Numerical results are reported to appropriate precision. Code is properly integrated with a tool like R Markdown or knitr, and both the knitted file and the source file are submitted. The code is indented, commented, and uses meaningful names. All code is relevant, without dangling or useless commands. All parts of all problems are answered with coherent sentences, and raw computer code or output are only shown when explicitly asked for.

[3]More specifically, `constrOptim()` handles linear inequality constraints, but that's all we need here. For more general constrained optimization, I like the `alabama` package, but there are many others.