FITTING MECHANISTIC MODELS TO EPIDEMIC CURVES VIA TRAJECTORY MATCHING

AARON A. KING

1. Likelihood: recapitulation

1. We focus on the *random process* that (putatively, hypothetically) *generated the data*
2. A model is just an explicit, mathematical description of this random process
3. The likelihood is (essentially) the *probability that the data were produced* given the model and the model's parameters
4. The likelihood quantifies (in some sense optimally) the model goodness of fit

Date: 21 May 2012.

The material in this document has been developed over the years with contributions from John M. Drake, Ben Bolker, Ottar Bjørnstad, Matt Ferrari, Helen Wearing, and David L. Smith.

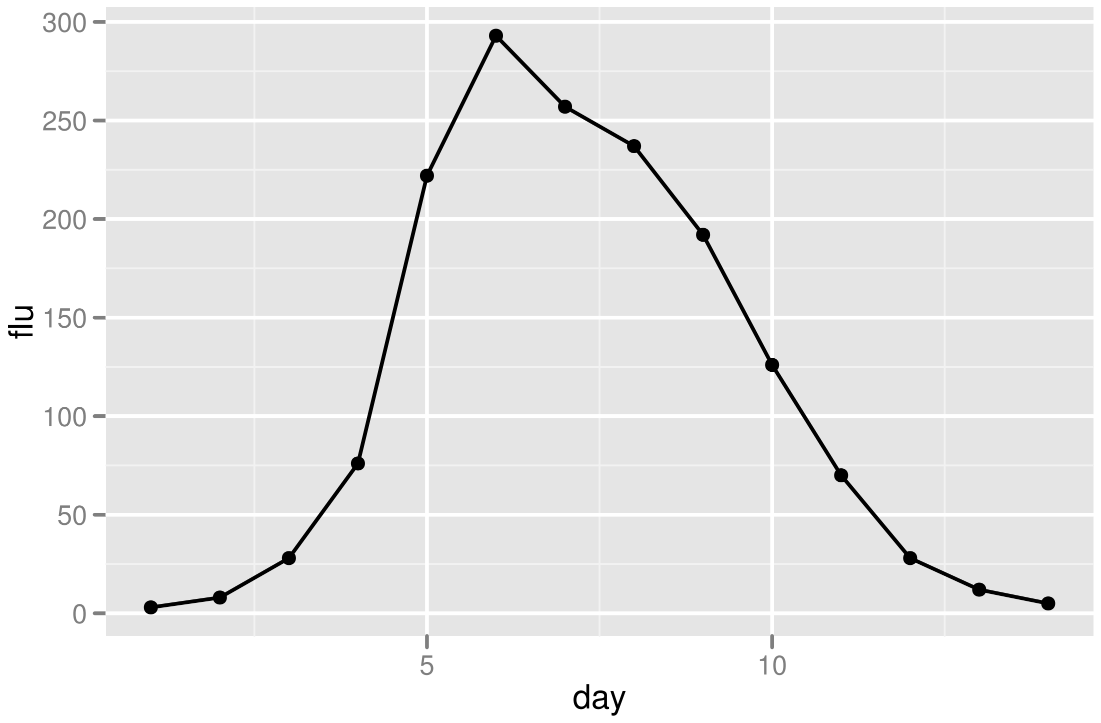


FIGURE 1. Daily number of schoolboys confined to bed by influenza during an outbreak in a British boarding school (Anonymous, 1978).

In this lecture, we'll take this point of view in an analysis of an epidemic curve (Fig. 1). That is, the data will be reported incidence through time. This is a specific sort of data, and, since we must model how the data were generated, we have to have a specific sort of model. However, the lecture will demonstrate the maximum likelihood approach to statistical inference, which is of great generality.

Load the data with

```
baseURL <- "http://www.math.mcmaster.ca/bolker/eeid/data"
flu <- read.csv(url(paste(baseURL,"boarding_school_flu.csv",sep="/")))
```

or

```
flu <- read.csv("boarding_school_flu.csv")
```

Plot it using

```
require(ggplot2)
ggplot(data=flu,mapping=aes(x=day,y=flu))+geom_point(size=2)+geom_line()
```

$$\begin{tikzcd}\boxed{S} \arrow[r, "\lambda"] & \boxed{I} \arrow[r, "\gamma"] & \boxed{B} \arrow[r, "\delta"] & \boxed{R}\end{tikzcd}$$

Figure 2. The SIR model of the spread of an immunizing infection through a population. All individuals within a box are assumed to be absolutely identical.

The SIR model. We'll first formalize an hypothesis about how the data were generated. That's another way of saying that we'll write a mathematical model of the process responsible for producing the data. Since the data were generated by an outbreak of an immunizing infection in a closed population, the closed SIR model is the simplest process capable of generating such an outbreak.

The data are numbers of kids confined to bed. These kids were presumably clinically infectious but effective not very infections, since their rate of contact with other kids was minimal. To capture this, we need to complicate the model a little bit by adding a class for kids confined to bed.

Fig. 2 shows a diagram of this SIBR model. The host population is divided into four classes according to their infection status: S, susceptible hosts; I, infected (and circulating) hosts; B, kids confined to bed; R, recovered and immune hosts. The S→I rate, λ, called the *force of infection*, depends on the number of infectious individuals according to $\lambda(t) = \beta\, I/N$. The I→B rate is γ; the B→R rate is δ.

The diagram above can be interpreted in numerous ways, both as a deterministic system and as a stochastic process. When viewed deterministically, the SIR model is a system of ordinary differential equations:

$$\begin{aligned}\frac{dS}{dt} &= -\lambda(I,t)\, S\\ \frac{dI}{dt} &= \lambda(I,t)\, S - \gamma\, I\\ \frac{dB}{dt} &= \gamma\, I - \delta\, B\\ \frac{dR}{dt} &= \delta\, B\end{aligned}$$

Here, S, I, B, and R are the *numbers* of individuals in each class. The so-called *force of infection*, λ, depends on the number of infectives, I. In particular, we'll assume that

$$\lambda(I,t) = \beta\, \frac{I}{N}$$

where β is the transmission rate and N is the total population size, so that the risk of infection a susceptible faces is proportional to the *prevalence* (the fraction of the population that is infected). We know that kids were confined to bed on average 3 da, so that $\delta \approx 1/3\ \text{da}^{-1}$. We don't know the transmission rate β, or γ, which corresponds to the amount of time individuals were infectious before being confined to bed. There is additionally some uncertainty regarding the initial number of infectives.

2. Solving ODEs in R

Like almost all epidemiological models, one can't solve these equations analytically. However, we can compute the trajectories of a continuous-time model such as this one by integrating the equations numerically. Doing this accurately involves a lot of calculation, and there are smart ways and not-so-smart ways of going about it. This very common problem has been very thoroughly studied by numerical analysts for generations so that, when the equations are smooth, well-behaved functions, excellent numerical integration algorithms are readily available to compute approximate solutions to high precision. In particular, R has several sophisticated ODE solvers which (for many problems) will give highly accurate solutions. These algorithms are flexible, automatically perform checks, and give informative errors and warnings. To use the numerical differential equation solver package, we load the deSolve package

```r
require(deSolve)
```

The ODE solver we'll use is called `ode`. (We can do `?ode` to learn about its many options.) `ode` needs to know the *initial values* of the state variables (`y`), the `times` at which we want solutions, the right-hand side of the ODE, `func`. The latter can optionally depend on some parameters (`parms`).

Let's encode the SIR model equations in a form suitable for use as the `func` argument to `ode`. To do this, we'll need to write a function:

```r
sibr.model <- function (t, x, params) {
  ## first extract the state variables
  S <- x[1]
  I <- x[2]
  B <- x[3]
  ## now extract the parameters
  beta <- params["beta"]
  gamma <- params["gamma"]
  delta <- params["delta"]
  N <- 763
  ## now code the model equations
  dS.dt <- -beta*S*I/N
  dI.dt <- beta*S*I/N-gamma*I
  dB.dt <- gamma*I-delta*B
  ## combine results into a single vector
  dxdt <- c(dS.dt,dI.dt,dB.dt)
  ## return result as a list!
  list(dxdt)
}
```

Note that the order and type of the arguments and output of this function must exactly match `ode`'s expectations. Thus, for instance, the time variable `t` must be the first argument even if, as is the case here, nothing in the function depends on time. [When the RHS of the ODE are independent of time, we say the ODE are *autonomous*.] Note also, that `ode` expects the values of the ODE RHS to be the first element of a `list`.

Now we can call `ode` to compute trajectories of the model. To do this, we'll need some values of the parameters:

```
params <- c(beta=2,gamma=1/3,delta=1/3)
```

Note that we've set the death and birth rates to zero, because we're looking at a *closed epidemic*. [Q: What is the infectious period of this disease? Q: What is R_0 in this case?]

We now state the times at which we want solutions and specify the *initial conditions*, i.e., the starting values of the state variables S, I, and B:

```
times <- seq(from=0,to=15,by=1/4) ## returns a sequence
xstart <- c(S=762,I=1,B=0)       ## initial conditions
```

Next, we compute a model trajectory with the `ode` command:

```
out <- ode(
           func=sibr.model,
           y=xstart,
           times=times,
           parms=params
           )
class(out)
## [1] "deSolve" "matrix"
head(out)
##      time     S     I       B
## [1,] 0.00 762.0 1.000 0.00000
## [2,] 0.25 761.4 1.516 0.09943
## [3,] 0.50 760.4 2.296 0.24215
## [4,] 0.75 759.0 3.475 0.45093
## [5,] 1.00 756.9 5.255 0.76003
## [6,] 1.25 753.7 7.931 1.22065
```

It's convenient to store the results in a data-frame:

```
out <- as.data.frame(
                     ode(
                         func=sibr.model,
                         y=xstart,
                         times=times,
                         parms=params
                         )
                     )
```

We plot the results using the commands:

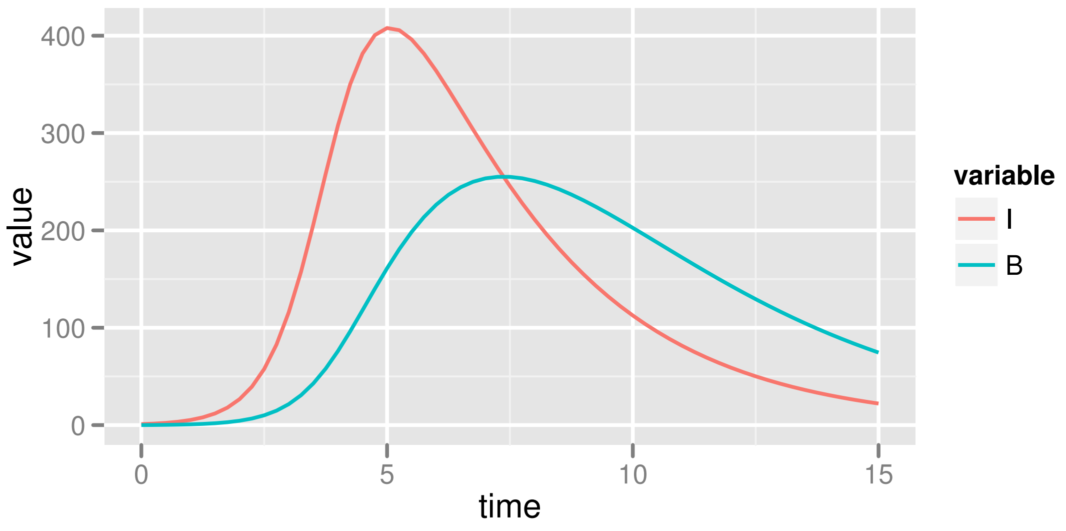


FIGURE 3. An SIBR outbreak in a closed population.

```r
require(ggplot2)
require(reshape)
out <- subset(out,select=c(I,B,time))
ggplot(data=melt(out,id.var="time"),
       mapping=aes(x=time,y=value,group=variable,color=variable))+
  geom_line()
```

Fig. 3 shows the result.

Exercise 1. Plot the trajectories of SIBR outbreaks for different values of the parameters β and γ.

Exercise 2. Reformulate the model to use $R_0 = \beta/\gamma$ and γ as parameters. Simulate to check that your code works as intended.

3. Estimating parameters using maximum likelihood

I've asserted that the SIBR model is the simplest model for this situation, and it is extremely simple. By itself, however, it's just too simple to be an explanation of any dataset. In particular, it is *deterministic*, which means that, if I know the states $S(t)$, $I(t)$, $B(t)$, at any time t, along with the parameters β, γ, and δ, then I know *everything* there is to know about the future of the system. Moreover, SIBR model predicts that epidemics will be smooth. Real data doesn't look like that. Put another way, if we're to view the SIR model as having generated the data, we need to add something to it to account for the deviations that exist between model predictions and the data.

Process noise vs. measurement error. Yesterday, we saw that there are two major categories of randomness in dynamic processes. Stochasticity in the process itself (e.g., due to variation in process rates, heterogeneity among individuals, random differences in the timing of discrete events, etc.) we term *process noise*. Yesterday, we saw how the chain binomial model can be used to capture this sort of stochasticity at a degree of model complexity comparable to that of the SIBR model above. On the other hand, *measurement error* reflects random errors made in the observation process itself (due to sampling, errors in recording, etc.). The feature that distinguishes these two sources of stochasticity is that measurement error has no effect on the real process (though obviously it degrades our inference), while process noise does. That is, a random perturbation to the process state at one time will alter the future trajectories of all the state variables.

Statistical inference using maximum likelihood is fairly straightforward as long as we consider models that have *either* process noise *or* measurement error, but not both. Dealing with the general situation requires considerably more careful treatment and is an active area of statistical research. I'll briefly indicate some recent directions forward for statistical inference on time series data when both measurement error and process noise are present later this afternoon.

Measurement error in the epidemic. Here, we'll explore the measurement-error-only assumption. In particular, we'll assume that the true process is entirely deterministic, and that an observation error at one time is independent of errors at other times. Maximum likelihood in this context is called *trajectory matching*.

Again, to use likelihood, we need a model that is capable of generating the data. In this case, the SIR equations predict the time evolution of the variables S, I, and R; we need to explicitly model the error to complete the model.

Let's assume that the errors are due to under-reporting. In particular, we might most naturally assume that reports C_t are binomially distributed:

$$C_t \;\sim\; \text{binomial}(B(t), p)$$

i.e, the *size* is the number of infectives at time t and p is the reporting probability. For various reasons, it's often better to consider a measurement model with more dispersion. In particular, the negative binomial model is often a better choice:

$$C_t \;\sim\; \text{negbin}(p\,B(t), k)$$

Here, the two parameters of the negative binomial distribution are the mean and the so-called "size". With these parameters, we have

$$\mathrm{Var}\,[C_t] = p\,B(t) + \frac{1}{k}\,(p\,B(t))^2 = B(t)\,p\,(1 + \frac{1}{k}\,p\,B(t))$$

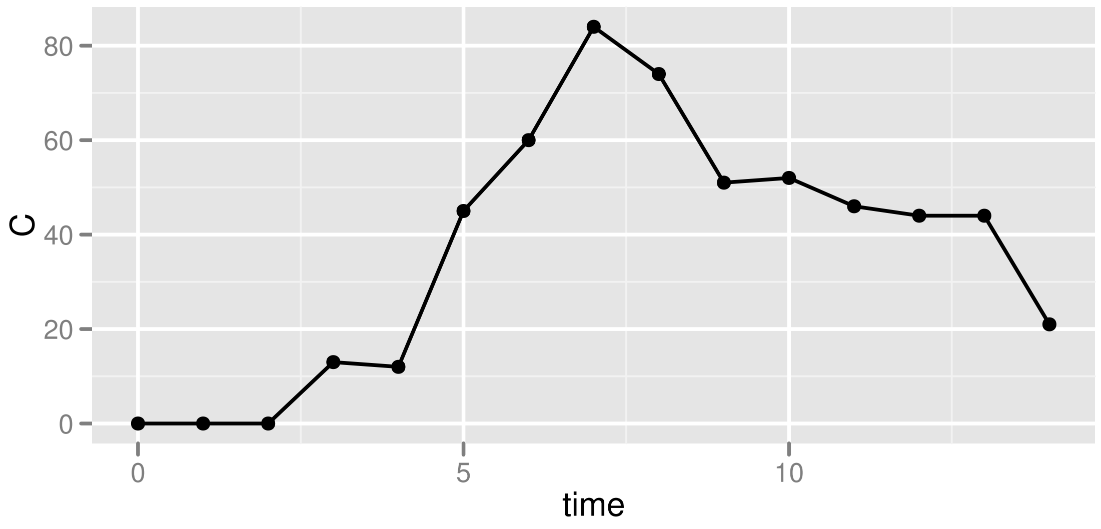
as compared with

$$\mathrm{Var}\,[C_t] = B(t)\,p\,(1-p)$$

under the binomial model.

To implement this in R, we can use the `nbinom` family of functions. The following codes simulate the full model:

```
times <- seq(from=0,to=14,by=1)
params <- c(beta=2,gamma=1/3,delta=1/3,N=763,p=0.3,k=1000)
out <- as.data.frame(
                     ode(
                         func=sibr.model,
                         y=xstart,
                         times=times,
                         parms=params
                         )
                     )
within(
       out,
       C <- rnbinom(n=length(B),mu=params["p"]*B,size=params["k"])
       ) -> out
ggplot(data=out,mapping=aes(x=time,y=C))+geom_point()+geom_line()
```



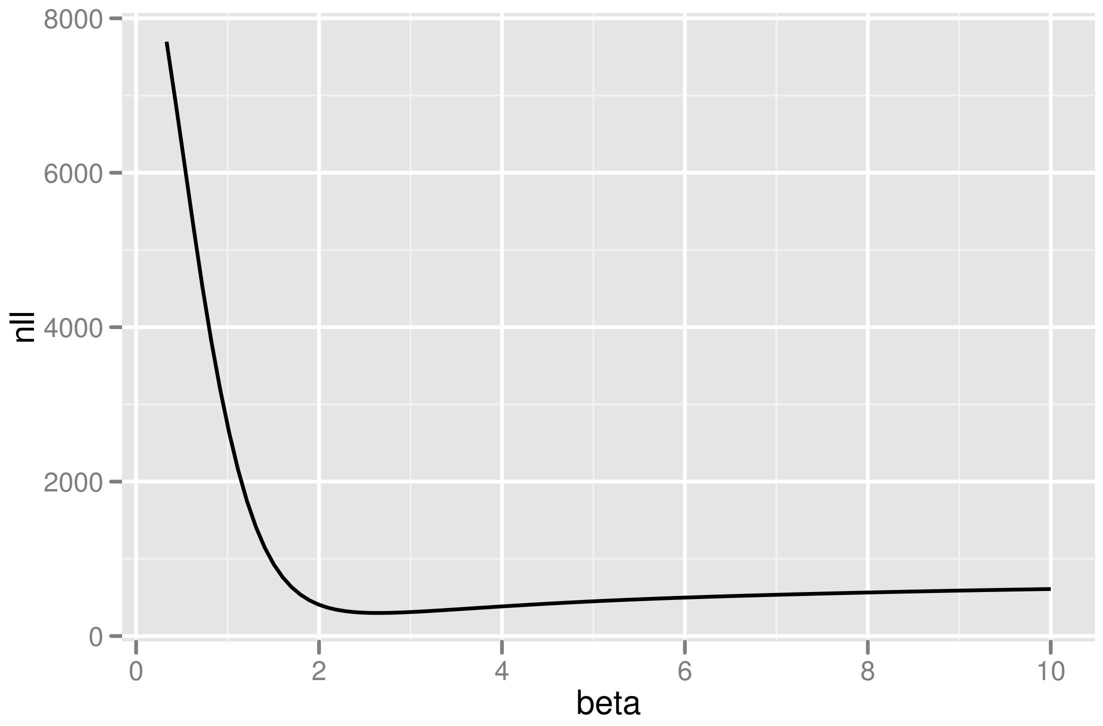
Exercise 3. Generate several simulated data sets for each of several different values of p and k.

The above codes *simulate* our model; to do statistical inference, we need to be able to *compute* the likelihood of the data given the model and some parameters. The following codes implement the negative log likelihood as a function of the model's parameters.

```
sibr.nll <- function (beta, gamma, delta, I.0, p, k) {
  times <- c(flu$day[1]-1,flu$day)
  ode.params <- c(beta=beta,gamma=gamma,delta=delta)
  xstart <- c(S=763-I.0,I=I.0,B=0)
  out <- ode(
             func=sibr.model,
             y=xstart,
             times=times,
             parms=ode.params
             )
  ## 'out' is a matrix
  ll <- dnbinom(x=flu$flu,size=params["k"],mu=p*out[-1,"B"],log=TRUE)
  -sum(ll)
}
```

We can use `optim` to find the parameter values that maximize the likelihood (actually, we'll equivalently be finding the parameters that minimize the negative log likelihood). Let's start small by estimating the single parameter β. The following codes plot the negative log likelihood as a function of β:

```
nll <- function (par) {
  sibr.nll(beta=par[1],gamma=1/3,delta=1/3,
             I.0=1,p=0.5,k=100)
}
betacurve <- data.frame(beta=seq(1/3,10,length=100))
within(betacurve,nll <- sapply(beta,nll)) -> betacurve
ggplot(data=betacurve,mapping=aes(x=beta,y=nll))+geom_line()
```

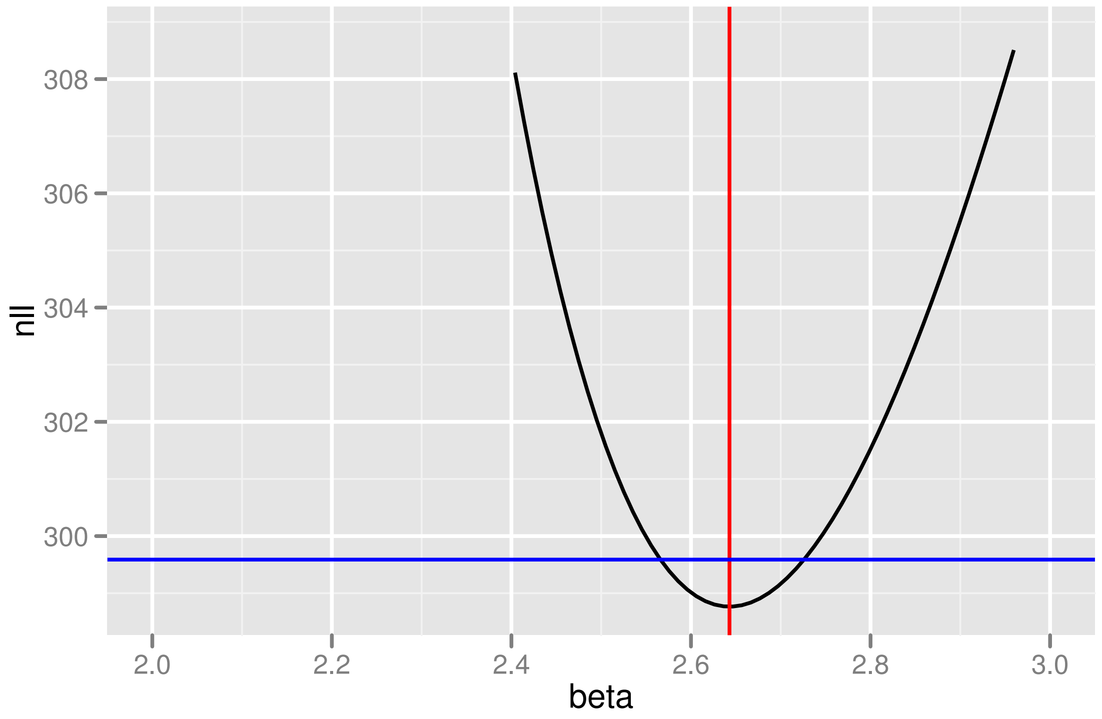


To find the exact value of the MLE for β, conditional on the assumptions we've made about the other parameters, we can use `optim`:

```
fit <- optim(fn=nll,par=2,method="Brent",lower=1.5,upper=3)
fit
## $par
## [1] 2.643
##
## $value
## [1] 298.8
##
## $counts
## function gradient
##       NA       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
```

We can use the likelihood-ratio theory we discussed yesterday to obtain a confidence interval for β (again, conditional on the assumed values of the other parameters):

```
crit.lr <- pchisq(q=0.05,df=1,lower.tail=FALSE)
betacurve <- data.frame(beta=seq(2,3,length=100))
betacurve <- within(betacurve,nll <- sapply(beta,nll))
ggplot(data=betacurve,mapping=aes(x=beta,y=nll))+geom_line()+
  ylim(fit$value+c(0,10))+
  geom_vline(xintercept=fit$par,color='red')+
  geom_hline(yintercept=fit$value+crit.lr,color='blue')
```

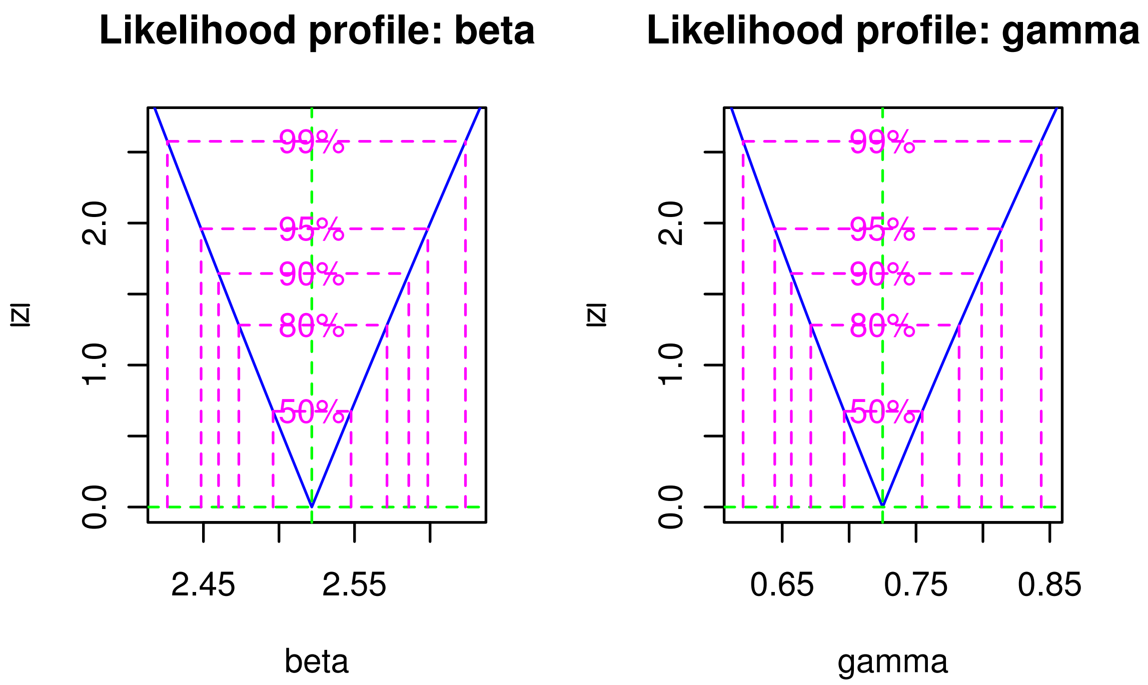


Now let's try and estimate more than one parameter.

```
nll <- function (par) {
  sibr.nll(beta=par[1],gamma=par[2],delta=1/3,
            I.0=1,p=0.5,k=100)
}
fit <- optim(fn=nll,par=c(2.6,1/3),method="Nelder-Mead")
fit
## $par
## [1] 2.522 0.725
##
## $value
## [1] 217.7
##
## $counts
## function gradient
##       53       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
```

This gives us the MLE, but for quantifying the uncertainty in more than one parameter, we'll need to do more than just compute the likelihood on slices through the parameter space. The bbmle package provides the useful mle2 command, which can isolate the MLE and compute profile likelihoods as well.

```
require(bbmle)
fit <- mle2(sibr.nll,start=list(beta=2.5,gamma=0.7),
            method="Nelder-Mead",
            fixed=list(delta=1/3,k=100,p=0.5,I.0=1))
coef(fit)
##     beta    gamma    delta      I.0        p        k
##   2.5217   0.7251   0.3333   1.0000   0.5000 100.0000
pfit <- profile(fit)
plot(pfit)
confint(pfit)
##        2.5 % 97.5 %
## beta  2.4485 2.5985
## gamma 0.6444 0.8139
```

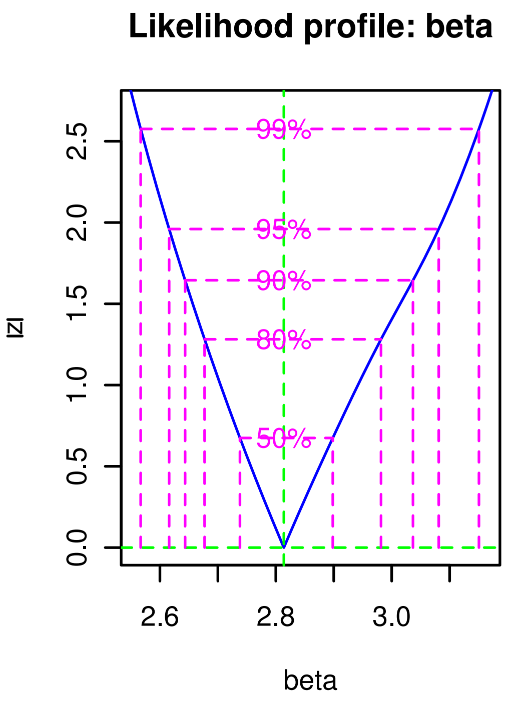


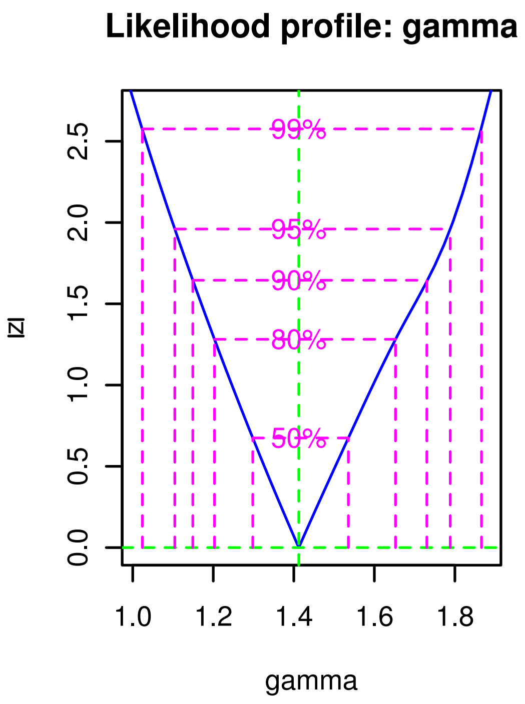
```
require(bbmle)
fit <- mle2(sibr.nll,
            start=list(beta=1.5,gamma=1/3,p=0.5),
            method="L-BFGS-B",
            lower=c(0,0,0),
            upper=c(Inf,Inf,1),
            fixed=list(delta=1/3,k=100,I.0=1))
coef(fit)
```

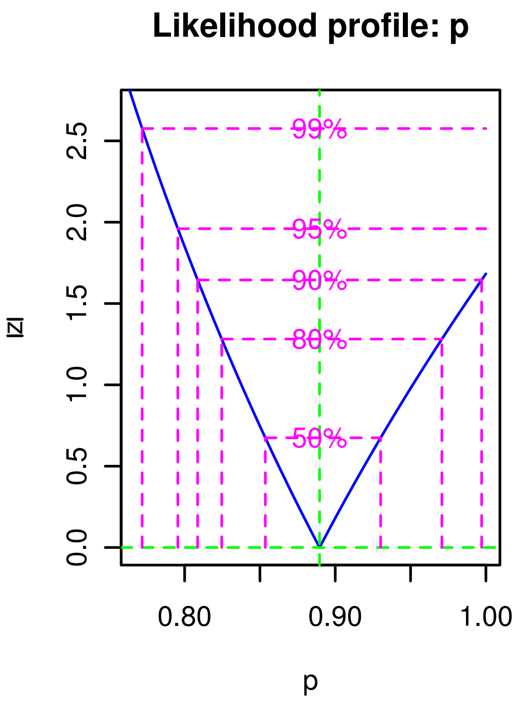
```
##     beta    gamma    delta      I.0        p        k
##   2.8139   1.4124   0.3333   1.0000   0.8896 100.0000
```

```
pfit <- profile(fit)
plot(pfit)
confint(pfit)
```

```
##        2.5 % 97.5 %
## beta  2.6163  3.081
## gamma 1.1048  1.787
## p     0.7956     NA
```





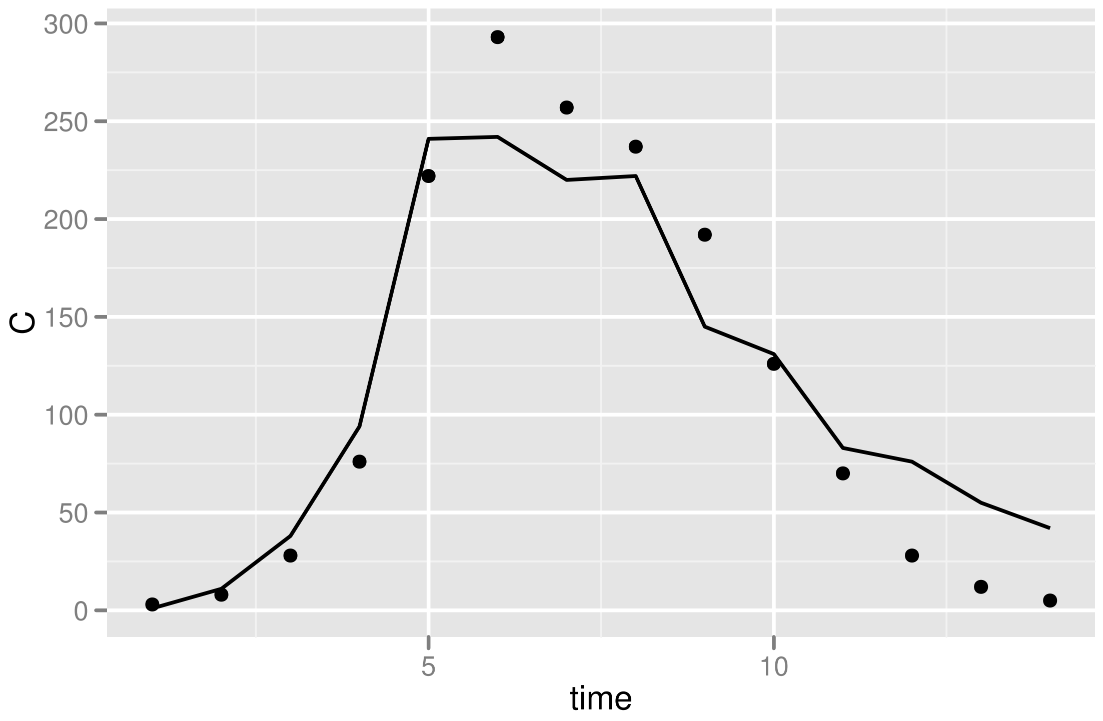


Let's check how well the model works by simulating from it at the MLE parameters.

```
mle <- coef(fit)
```

```
times <- seq(from=0,to=14,by=1)
xstart <- c(S=763-mle["I.0"],I=mle["I.0"],B=0)
out <- as.data.frame(
                     ode(
                         func=sibr.model,
                         y=xstart,
                         times=times,
                         parms=mle
                         )
                     )
within(
       subset(out,time>0),
       C <- rnbinom(n=length(B),mu=mle["p"]*B,size=mle["k"])
       ) -> out

ggplot(data=out,mapping=aes(x=time,y=C))+geom_line()+
  geom_point(data=flu,mapping=aes(x=day,y=flu))
```



Exercise 4. Try fitting additional parameters. The overdispersion parameter, k, and the initial number of infectives, $I(0)$, seem the most interesting choices.

Exercise 5. Try formulating a model for the Bombay plague mortality data:

```
plague <- read.csv(url(paste(baseURL,"bombay_plague.csv",sep="/")))
## Error: cannot open the connection
```

Estimate parameters using trajectory matching.

References

Anonymous. 1978. Influenza in a boarding school. British Medical Journal, **1**:587.

A. A. King, Departments of Ecology & Evolutionary Biology and Mathematics, University of Michigan, Ann Arbor, Michigan 48109-1048 USA
E-mail address: `kingaa at umich dot edu`
URL: `http://kinglab.eeb.lsa.umich.edu`