Introduction to model parameter estimation

Aaron A. King
with contributions from
Ottar Bjørnstad, Ben Bolker, John Drake, Pej Rohani, and Dave Smith

October 30, 2017

1 Introduction

This course will focus on the utility of models in understanding, predicting, and controlling infectious disease systems. Models play a central role in this work because they allow us to precisely and quantitatively express our ideas about the mechanisms of infectious disease transmission, immunity, and ecology. To the extent our understanding of the important mechanisms is correct, such models are extremely useful in the design of policy. On the other hand, models are useful in uncovering the important mechanisms, inasmuch as we can compare model predictions to data. Specifically, if we can translate competing hypotheses into mathematical models, these can be compared in terms of their ability to correctly capture the patterns seen in data. In order to fairly compare competing models, we must first try to find out what is the best each model can do. Practically speaking, this means that we have to find the values of the models' parameters that give the closest correspondence between model predictions and data. Parameter estimation can be important even when we are fairly confident in the ability of a single model to explain the dynamics. Not surprisingly, the all-important quantity R_0 is frequently the focus of considerable parameter-estimation effort. Here, we'll try our hand as estimating R_0 and other model parameters from an epidemic curve using a couple of different methods.

2 Estimating R_0 in an invasion

We saw in the lecture how, during the early stages of an outbreak, the number of infected individuals Y is approximately

$$Y \approx Y_0 e^{((R_0-1)(\gamma+\mu)t)}$$

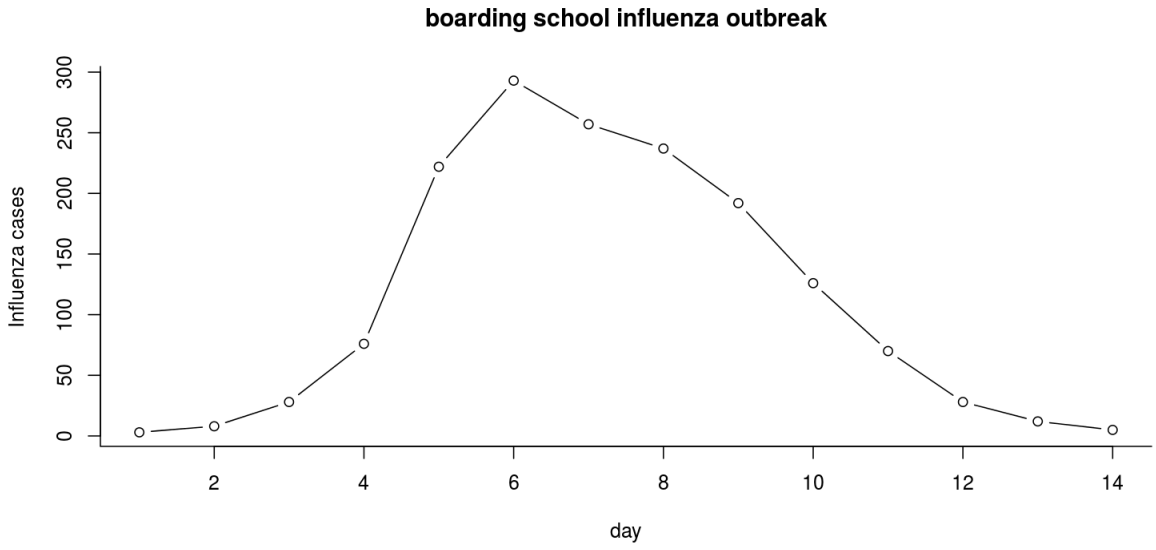
where Y_0 is the (small) number of infectives at time 0, $\frac{1}{\gamma}$ is the infectious period, and $\frac{1}{\mu}$ is the host lifespan. Taking logs of both sides, we get

$$\log Y \approx \log Y_0 + (R_0 - 1)(\gamma + \mu)t,$$

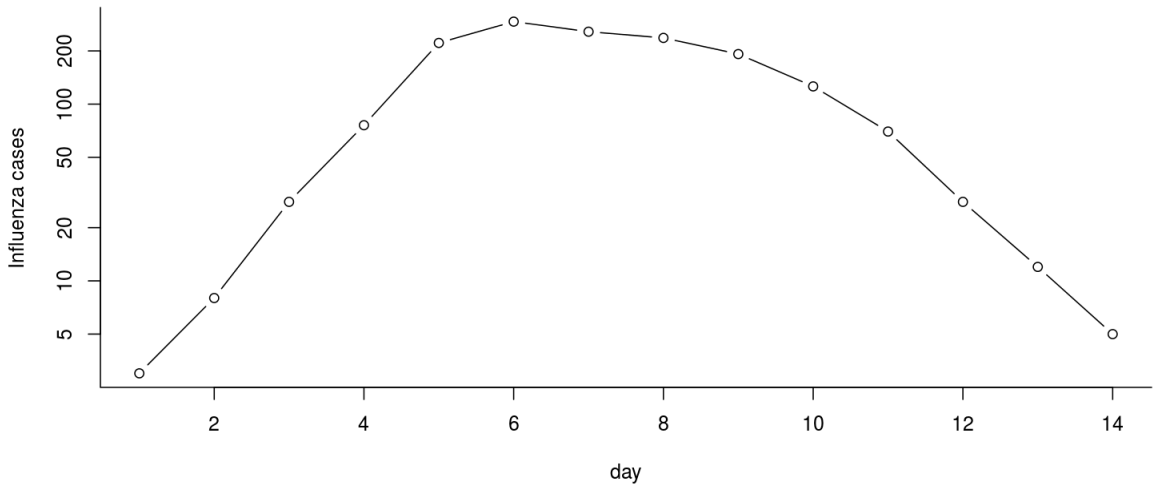
which implies that a semi-log plot of Y vs t should be approximately linear with a slope proportional to $R_0 - 1$ and the recovery rate.

We can plot the 1977 boarding school influenza data in this way to see if this is the case.

```
url <- "https://kingaa.github.io/thid/data/bbs.csv"
flu <- read.csv(url)
## or download the file and do:
## flu <- read.csv(file="bbs.csv")
plot(flu~day,data=flu,type='b',bty='l',
     main='boarding school influenza outbreak',
     xlab='day',ylab='Influenza cases')
```




```
plot(flu~day,data=flu,type='b',log='y',bty='l',
     xlab='day',ylab='Influenza cases')
```



Plotted on a log scale, the linearity of the first several data points is indeed striking. This suggests that we can obtain a cheap and cheerful estimate of R_0 by a simple linear regression.

```
fit <- lm(log(flu)~day,data=subset(flu,day<=4))
summary(fit)

##
## Call:
## lm(formula = log(flu) ~ day, data = subset(flu, day <= 4))
##
## Residuals:
##        1        2        3        4
##  0.03073 -0.08335  0.07450 -0.02188
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.02703    0.10218  -0.265  0.81611
## day          1.09491    0.03731  29.346  0.00116 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08343 on 2 degrees of freedom
## Multiple R-squared:  0.9977,Adjusted R-squared:  0.9965
## F-statistic: 861.2 on 1 and 2 DF,  p-value: 0.001159

coef(fit)

## (Intercept)         day
## -0.02703361  1.09491261
```

```
slope <- coef(fit)[2]
slope
```

```
##      day
## 1.094913
```

Now, we know that influenza's infectious period is about 2.5 da. Moreover, since this is far shorter than an average human life ($\mu \ll \gamma$), we can neglect μ in our estimating equation. Thus our estimate of R_0 is

$$\hat{R_0} = \texttt{slope}/\gamma + 1 \approx 1.1 \times 2.5 + 1 \approx 3.7.$$

Our strategy in this case has been to redefine the problem so that it fits a standard form, i.e., we showed how to rearrange the model so that the relevant quantity (R_0) could be obtained from linear regression. We can get a rough estimate of the uncertainty in our estimate by looking at the standard errors in our estimator.

```
coef(summary(fit))
```

```
##                Estimate Std. Error   t value    Pr(>|t|)
## (Intercept) -0.02703361 0.10217980 -0.264569 0.816111672
## day          1.09491261 0.03731079 29.345738 0.001159189
```

```
slope.se <- coef(summary(fit))[2,2]
2.5*slope.se
```

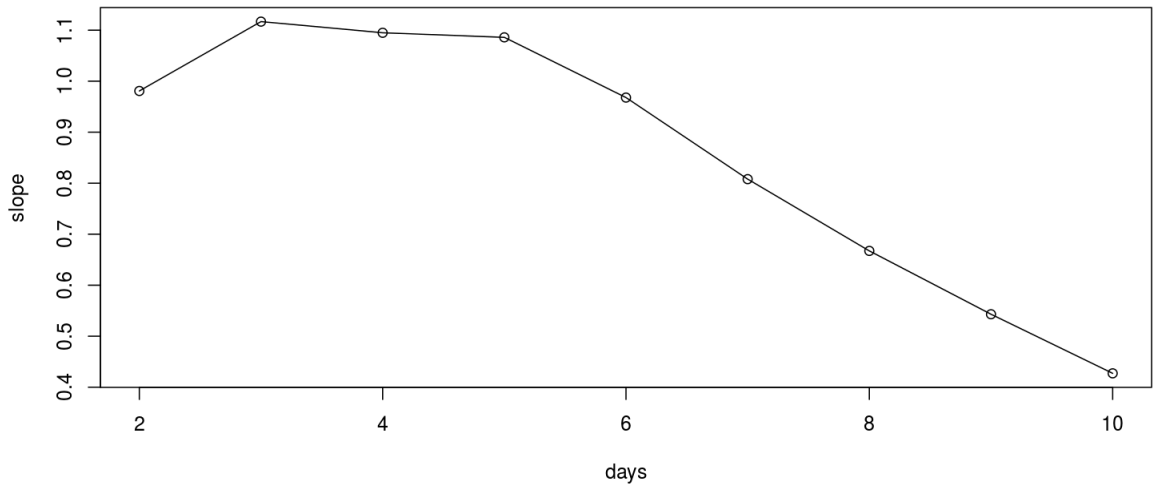
```
## [1] 0.09327697
```

So we reckon we've got an error of ± 0.09 in our estimate of R_0, i.e., we feel pretty confident that $3.55 < R_0 < 3.92$.

A defect of this method is that it uses only a small amount of the data to compute an important quantity. Moreover, we have to make a subjective judgement as to how much of the data to use. Further, as we use more data, and presumably obtain more precise estimates, we simulataneously get further from the realm where our approximation is valid, which introduces greater bias. Let's see how our estimates of R_0 depend on what we choose to be the "initial phase" of the outbreak. Below, we estimate R_0 and its standard error using the first 2, 3, 4, ..., 10 data points.

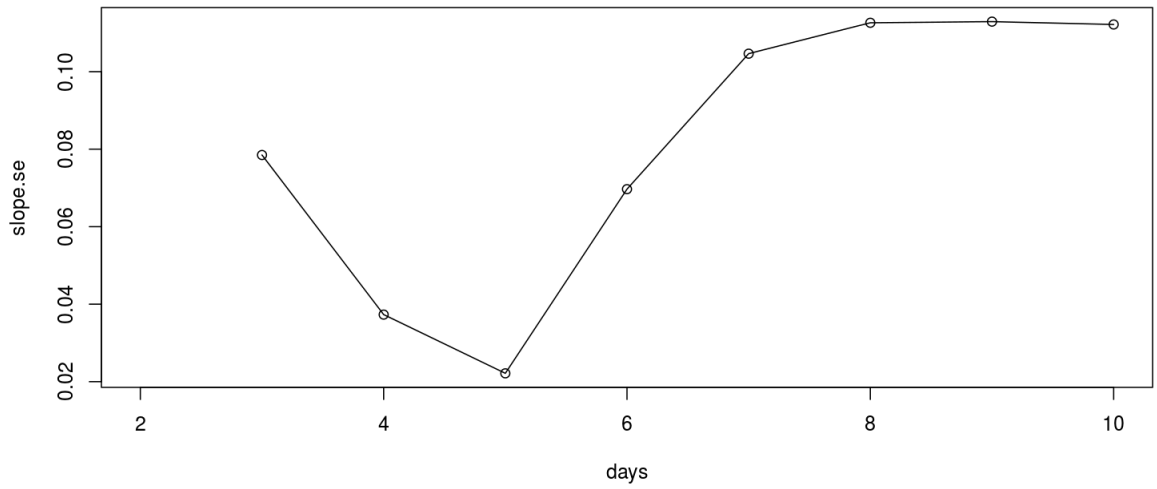
```
days <- 2:10
slope <- numeric(length=length(days))
slope.se <- numeric(length=length(days))
```

```r
for (k in seq_along(days)) {
  fit <- lm(log(flu)~day,data=subset(flu,day<=days[k]))
  slope[k] <- coef(summary(fit))[2,1]
  slope.se[k] <- coef(summary(fit))[2,2]
}
R0.hat <- slope*2.5+1
R0.se <- slope.se*2.5
plot(slope~days,type='o')
```

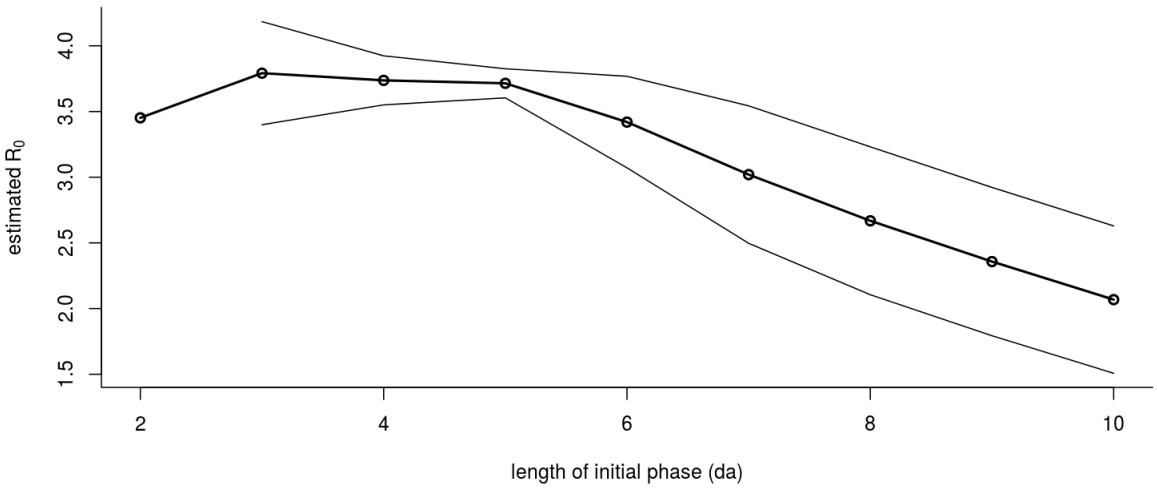


We'll plot these estimates against their uncertainties to show this precision-accuracy tradeoff.

```r
plot(slope.se~days,type='o')
```




```
plot(range(days),
     range(c(R0.hat-2*R0.se,R0.hat+2*R0.se),na.rm=T),
     type='n',bty='l',
     xlab="length of initial phase (da)",
     ylab=expression("estimated"~R[0]))
lines(R0.hat~days,type='o',lwd=2)
lines(R0.hat+2*R0.se~days,type='l')
lines(R0.hat-2*R0.se~days,type='l')
```



Exercise 1. Biweekly data for outbreaks of measles in three communities within Niamey, Niger are provided in the file https://kingaa.github.io/thid/data/niamey.

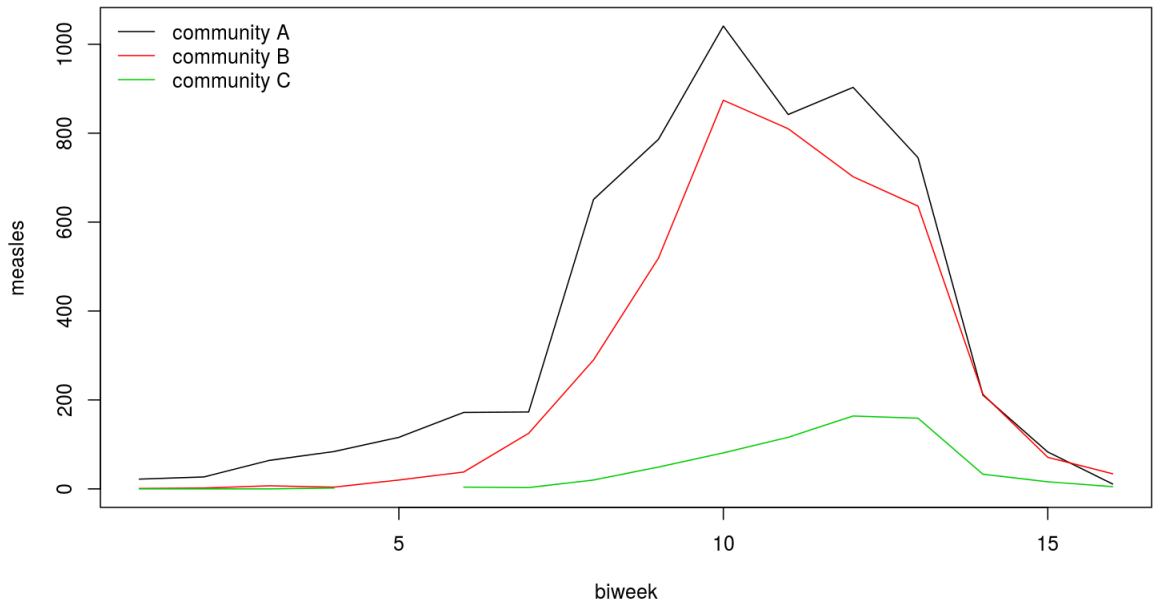
csv. Use this method to obtain estimates of R_0 for measles using the data from each of the communities of Niamey, assuming that the infectious period is approximately two weeks.

3 Fitting deterministic dynamical epidemiological models to data

Now we move on to a much more general but considerably more complicated technique for estimating R_0. The method of *least squares* gives us a way to quantify the discrepancy between the data and a model's predictions. We can then search over all possible values of a model's parameters to find the parameters that minimize this discrepancy.

We'll illustrate this method using the Niamey data, which we'll load and plot using the following commands:

```
url <- "https://kingaa.github.io/thid/data/niamey.csv"
niamey <- read.csv(url)
plot(measles~biweek,data=niamey,type='n')
lines(measles~biweek,data=subset(niamey,community=="A"),col=1)
lines(measles~biweek,data=subset(niamey,community=="B"),col=2)
lines(measles~biweek,data=subset(niamey,community=="C"),col=3)
legend("topleft",col=1:3,lty=1,bty='n',
       legend=paste("community",c("A","B","C")))
```



Since this is a single outbreak, and the number of births and deaths into the population over the course of the outbreak is small relative to the size of the population, we can treat this outbreak as if it were occurring in a closed population. We saw earlier how we can formulate the SIR equations in such a case and how we can solve the model equations to obtain trajectories. Before, we formulated the model in terms of the *fractions*, S, I, R of the host population in each compartment. Here, however, we need to track the *numbers*, X, Y, Z, of individuals in the S, I, and R compartments, respectively. In terms of X, Y, Z, our frequency-dependent SIR model is

$$\begin{aligned}\frac{dX}{dt} &= -\frac{\beta X Y}{N}\\ \frac{dY}{dt} &= \frac{\beta X Y}{N} - \gamma Y\\ \frac{dZ}{dt} &= \gamma Y\end{aligned}$$

Where $N = X + Y + Z$ is the total host population size. A function suitable for use with `deSolve` for the closed SIR epidemic is:

```
require(deSolve)

closed.sir.model <- function (t, x, params) {
  X <- x[1]
  Y <- x[2]
  Z <- x[3]

  beta <- params["beta"]
  gamma <- params["gamma"]
  pop <- params["popsize"]

  dXdt <- -beta*X*Y/pop
  dYdt <- beta*X*Y/pop-gamma*Y
  dZdt <- gamma*Y

  list(c(dXdt,dYdt,dZdt))
}
```

Thus far, we have only considered deterministic models. In the next lab, we will begin to think about more realistic models that begin to take into account some aspects of the stochastic nature of real epidemics. For now, under the assumption that the epidemic is deterministic, parameter estimation is a matter of finding the model trajectory that gives the best fit to the data. The first thing we need is a function that computes a trajectory given parameters of the model.

```
prediction <- function (params, times) {
  xstart <- params[c("X.0","Y.0","Z.0")]
  out <- ode(
             func=closed.sir.model,
```

```
              y=xstart,
              times=times,
              parms=params
              )
  out[,3]      # return the number of infectives
}
```

Now we set up a function that will calculate the sum of the squared differences (or errors) between the data and the model predictions.

```
sse <- function (params, data) {
  times <- c(0,data$biweek/26)           # convert to years
  pred <- prediction(params,times)
  discrep <- pred[-1]-data$measles
  sum(discrep^2)                         # sum of squared errors
}
```

To get a sense of what this gives us, let's explore varying some parameters and computing the SSE for community "A" of the Niamey data set. To begin with, we'll assume we know that $\gamma = 365/13$ and that the initial numbers of susceptibles, infectives, and recovereds, X_0, Y_0, Z_0 were 10000, 10, and 20000, respectively. We'll write a little function that will plug a value of β into the parameter vector and compute the SSE.

```
dat <- subset(niamey,community=="A")
params <- c(X.0=10000,Y.0=10,Z.0=39990,popsize=50000,
            gamma=365/13,beta=NA)
f <- function (beta) {
  params["beta"] <- beta
  sse(params,dat)
}
beta <- seq(from=0,to=1000,by=5)
SSE <- sapply(beta,f)
```

We take our estimate, $\hat{\beta}$ to be the value of β that gives the smallest SSE.

```
beta.hat <- beta[which.min(SSE)]
```

Fig. 1A shows SSE vs. β. What does the SIR model predict at $\beta = \hat{\beta}$? We compute the model's trajectory to find out:

```
params["beta"] <- beta.hat
plot(measles~biweek,data=dat)
lines(dat$biweek,prediction(params,dat$biweek/26))
```

This plot is Fig. 1B.

Exercise 2. Use this method to obtain estimates of R_0 for measles from each of the three communities in Niamey. You may again assume that the infectious period is approximately two weeks.

Clearly, this fit leaves much to be desired, but recall that we've here assumed that we know the correct values for all parameters but β. In particular, we've assumed we know the infectious period, and the initial conditions. [NB: the initial value of Z is entirely irrelevant. Why?] Let's see what happens when we try to estimate two parameters at once.

```
dat <- subset(niamey,community=="A")
params <- c(X.0=NA,Y.0=10,Z.0=1,popsize=50000,
            gamma=365/13,beta=NA)
f <- function (par) {
  params["beta"] <- par[1]
  params["X.0"] <- par[2]
  sse(params,dat)
}
beta <- seq(from=100,to=300,by=5)
X.0 <- seq(from=4000,to=20000,by=1000)
grid <- expand.grid(beta=beta,X.0=X.0)
grid$SSE <- apply(grid,1,f)
```

We can visualize this as a surface. A convenient function for this is `contourplot` from the lattice package:

```
require(lattice)
contourplot(sqrt(SSE)~beta+X.0,data=grid,cuts=30)
```

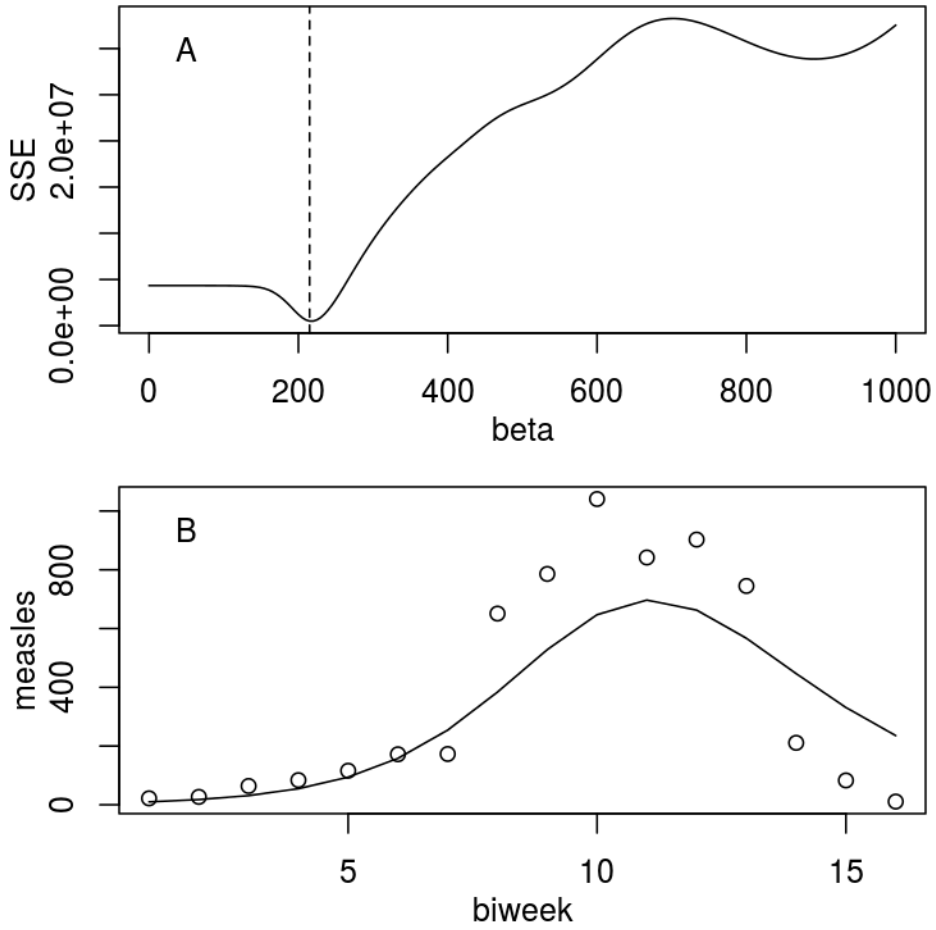
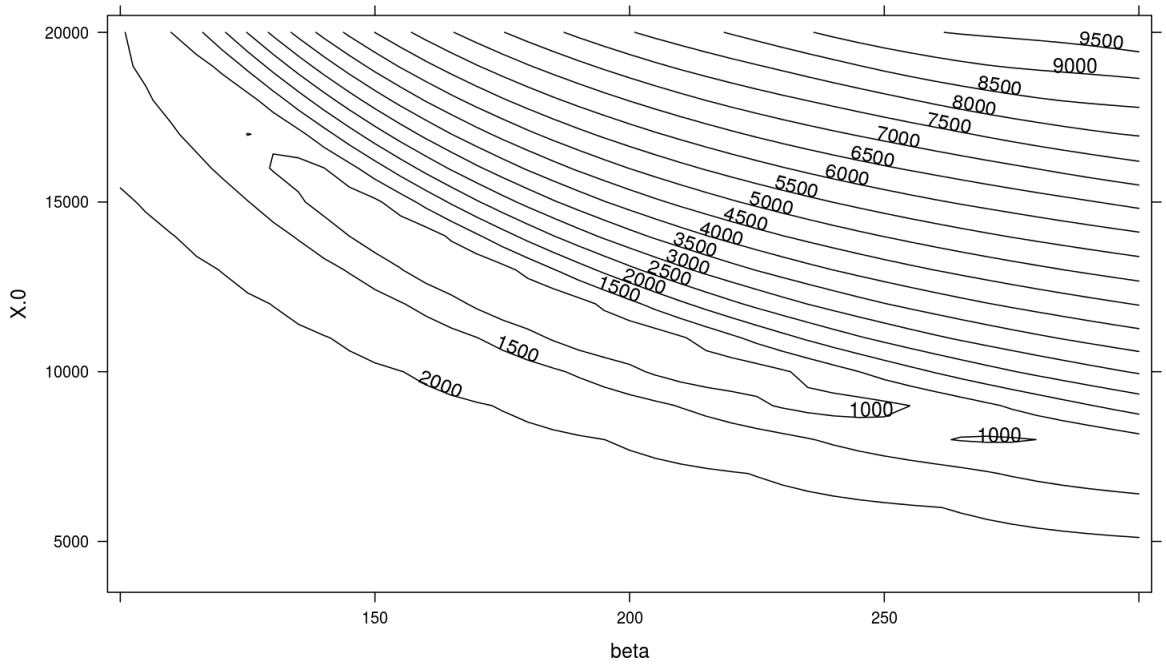


Figure 1: (A) Discrepancy (sum of squared error) between SIR model predictions and the measles data from Niamey community "A" as a function of β. The vertical line gives the least-squared estimate, $\hat{\beta} = 220$. (B) Trajectory of the model at $\beta = \hat{\beta}$ (solid curve) compared to the data (circles).



Exercise 3. Discuss the shape of this surface: what does it tell us about the uncertainty in the model's parameters?

***Exercise 4.** Repeat the estimation using a closed SEIR model. Assume that the infectious period is 5 da and the latent period is 8 da. How and why does your estimate of R_0 differ from that you obtained using the SIR model?

4 Optimization algorithms

When we have more than two parameters to estimate (as we usually will), we cannot rely on grid searches or simple graphical techniques to find the region of parameter space with the best parameters. We need more systematic ways of searching through the parameter space. Mathematicians have devoted much study to *optimization algorithms*, and there are many of these. Many of them are implemented in R.

The first place to go is the function `optim`, which implements several common, well-studied, generally-useful optimization algorithms.

```
?optim
```

To use it, we have to specify the function we want to *minimize* and a starting value for the parameters. Starting from this point, `optim`'s algorithms will search the parameter space for the value that minimizes the value of our *objective function*.

We'll write an objective function to try to estimate β, X_0, and Y_0 simultaneously. For the moment, we'll continue to assume that the recovery rate γ is known.

```
dat <- subset(niamey,community=="A")
params <- c(X.0=NA,Y.0=NA,Z.0=1,popsize=50000,
            gamma=365/13,beta=NA)
f <- function (par) {
  params[c("X.0","Y.0","beta")] <- par
  sse(params,dat)
}
optim(fn=f,par=c(10000,10,220)) -> fit
```

```
fit
```

```
## $par
## [1] 9150.769987    1.017659  271.651714
##
## $value
## [1] 166584.5
##
## $counts
## function gradient
##      273       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Exercise 5. In the foregoing, we've estimated parameters by minimizing the sum of squared differences between model-predicted number of cases and the data. What would happen if we tried to minimize the squared error on the log scale, i.e., to minimize $(\log(\text{model}) - \log(\text{data}))^2$? What would happen if we minimized the squared error on the square-root scale, i.e., $(\sqrt{\text{model}} - \sqrt{\text{data}})^2$? What's the "right" scale to choose?

***Exercise 6.** Try to estimate all four parameters at once. Start your algorithm from several places to check that they all converge to the same place. You may find it useful to restart the optimizer to verify its convergence.

***Exercise 7.** Fig. 1 shows a second local minimum of the SSE at a much higher value of β. Why is this?

Many other optimization algorithms exist. `optim` implements several of these (see `?optim`). Other functions and packages you might look into include: `constrOptim`, `optimx`, `nlm`, `nlminb`, `nloptr` (from the nloptr package), and `subplex` (from the subplex package).

5 The likelihood

We have seen that fitting mechanistic models to data is a powerful and general approach to estimating parameters. We saw too that least-squares fitting, is a straightforward way to do this. However, several issues arose. First, there was an element of arbitrariness in the choice of discrepancy measure. Second, although we could fairly easily obtain point estimates of model parameters using least-squares, it was not clear how we could obtain concomitant estimates of parameter uncertainty (e.g., confidence intervals). Finally, we began to see that there are limits to our ability to estimate parameters. In this lab, we'll explore these issues, and see that likelihood offers an attractive resolution to the first and second of these, but that the third is a fundamental challenge.

Likelihood has many advantages:

1. fidelity to model
2. a deep and general theory
3. a sound theoretical basis for confidence intervals and model selection
4. statistical efficiency

and some disadvantages:

1. fidelity to model
2. fragility (lack of robustness)

General definition

Likelihood is the *probability of a given set of data D having occurred under a particular hypothesis H*:

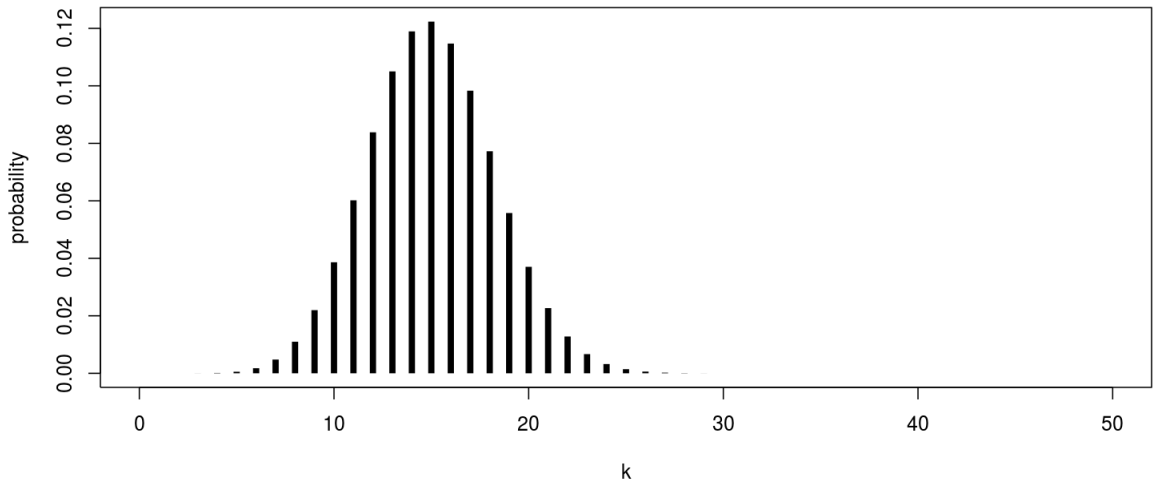
$$\mathcal{L}(H, D) = \mathbb{P}\left[D|H\right]$$

A simple example: suppose n individuals participate in a serological survey and k of these individuals are found to be seropositive. One parameter of interest is the true fraction, p, of the population that has seroconverted. Assuming the sample was drawn at random and the population is large, then the probability of the data (m of n individuals seropositive) given the hypothesis that the true probability is p is

$$\mathbb{P}\left[D|H\right] = \binom{n}{k} p^k \left(1-p\right)^{n-k}$$

If the true seroprevalence was, say, $p = 0.3$, what does the probability of observing k seropositives in a sample of size $n = 50$ look like?

```r
p <- 0.3
n <- 50
k <- seq(0,50,by=1)
prob <- dbinom(x=k,size=n,prob=p)
plot(k,prob,type='h',lwd=5,lend=1,
     ylab="probability")
```

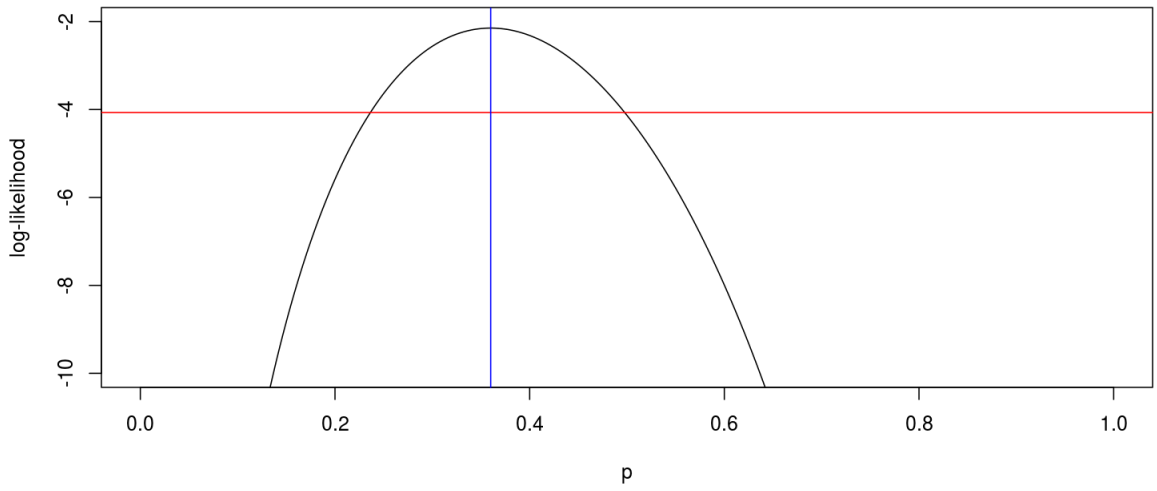


The likelihood is a function of the unknown parameters. In this case, if we assume n is known, then the likelihood is a function of p alone:

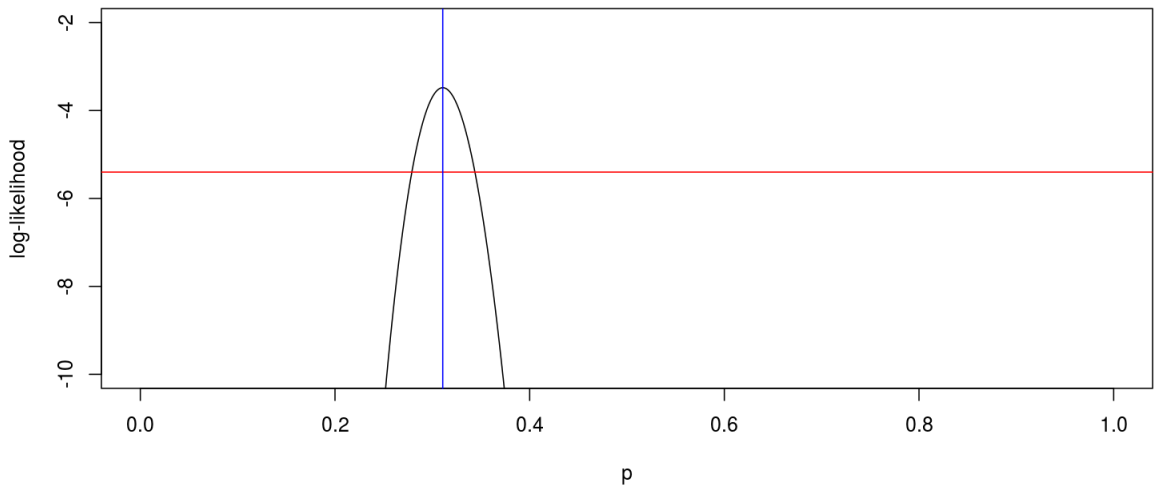
$$\mathcal{L}(p) = \binom{n}{k} p^k (1-p)^{n-k}$$

Typically the logarithm of this function is more interesting than $\mathcal{L}$ itself. Looking at this function for each of two different surveys:

```r
k1 <- 18
n1 <- 50
p <- seq(0,1,by=0.001)
plot(p,dbinom(x=k1,size=n1,prob=p,log=TRUE),
     ylim=c(-10,-2),ylab="log-likelihood",
     type='l')
abline(h=dbinom(x=k1,size=n1,prob=k1/n1,log=TRUE)-
       0.5*qchisq(p=0.95,df=1),col='red')
abline(v=k1/n1,col='blue')
```




```
k2 <- 243
n2 <- 782
p <- seq(0,1,by=0.001)
plot(p,dbinom(x=k2,size=n2,prob=p,log=TRUE),
     ylim=c(-10,-2),ylab="log-likelihood",
     type='l')
abline(h=dbinom(x=k2,size=n2,prob=k2/n2,log=TRUE)-
       0.5*qchisq(p=0.95,df=1),col='red')
abline(v=k2/n2,col='blue')
```



In the above two plots, the likelihood is a function of the model parameter p. Vertical lines show

the maximum likelihood estimate (MLE) of p. Horizontal lines show the critical likelihoods for the likelihood ratio test at the 95% confidence level.

Exercise 8. How do the two curves just plotted differ from one another? What features of the data are responsible for the differences?

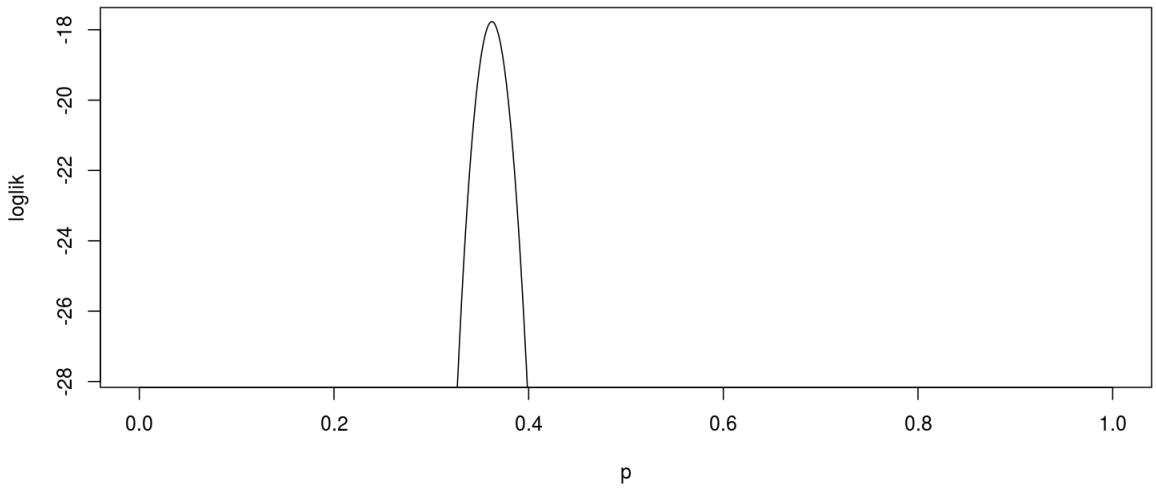
From data points to data sets

Let's suppose we have three samples, D_1, D_2, D_3, taken by three different researchers, for the same large population. If these samples are *independent*, then

$$\mathbb{P}[D|H] = \mathbb{P}[D_1|H] \times \mathbb{P}[D_2|H] \times \mathbb{P}[D_3|H]$$

which means that the likelihood of the full data set is the product of the likelihoods from each of the samples. In other words, the likelihood gives a general recipe for combining data from different studies. We'd compute the likelihood as follows:

```
n <- c(13,484,3200)
k <- c(4,217,1118)
dbinom(x=k,size=n,prob=0.2,log=TRUE)

## [1]   -1.873761  -79.243371 -197.561806

sum(dbinom(x=k,size=n,prob=0.2,log=TRUE))

## [1] -278.6789

ll.fn <- function (p) {
  sum(dbinom(x=k,size=n,prob=p,log=TRUE))
}
p <- seq(0,1,by=0.001)
loglik <- sapply(p,ll.fn)
plot(p,loglik,type='l',ylim=max(loglik)+c(-10,0))
```



6 Fitting SIR to an epidemic curve using likelihood

Let's revisit the model-fitting we did yesterday for the case of measles in Niger. We'll simplify the model slightly to eliminate some unnecessary and wasteful elements. Our frequency-dependent SIR model, again, is

$$\frac{dX}{dt} = -\frac{\beta X Y}{N}$$
$$\frac{dY}{dt} = \frac{\beta X Y}{N} - \gamma Y$$
$$\frac{dZ}{dt} = \gamma Y$$

Notice that 1. the Z equation is irrelevant for the dynamics of the epidemic and we can drop it entirely, and 2. β only ever occurs in combination with N, so we can combine these two into a single parameter by defining $b = \beta/N$. We can modify the R codes we used before to take account of this.

```
require(deSolve)

closed.sir.model <- function (t, x, params) {
  inc <- params["b"]*x[1]*x[2]           # incidence
  list(c(-inc,inc-params["gamma"]*x[2]))
}
```

```
prediction <- function (params, times) {
  out <- ode(
              func=closed.sir.model,
              y=params[c("X.0","Y.0")],
              times=c(0,times),
              parms=params
              )
## return the Y variable only
## and discard Y(0)
  out[-1,3]
}
```

Earlier, we used SSE as a measure of the discrepancy between model predictions and data:

```
sse <- function (params, data) {
  times <- data$biweek/26                # convert to years
  pred <- prediction(params,times)
  discrep <- pred-data$measles
  sum(discrep^2)                         # sum of squared errors
}
```

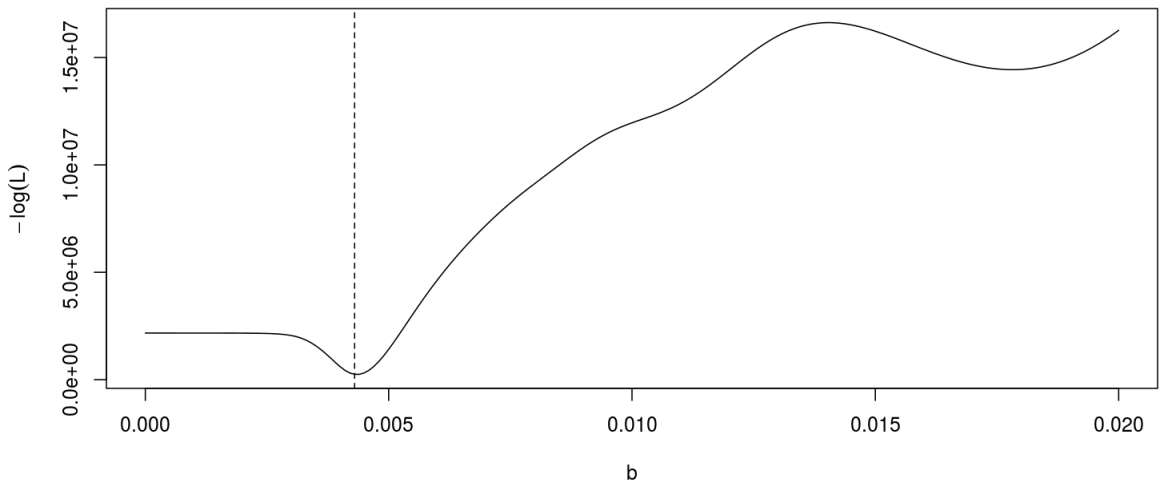
Now let's use likelihood instead. Let's suppose that, when we record cases, we make errors that are normal. Here's how we can compute the likelihood of the data given the model and its parameters:

```
loglik <- function (params, data) {
  times <- data$biweek/26
  pred <- prediction(params,times)
  sum(dnorm(x=data$measles,mean=pred,sd=params["sigma"],log=TRUE))
}
```

```
dat <- subset(niamey,community=="A")
params <- c(X.0=10000,Y.0=10,gamma=365/13,b=NA,sigma=1)

f <- function (b) {
  par <- params
  par["b"] <- b
  loglik(par,dat)
}

b <- seq(from=0,to=0.02,by=0.0001)
ll <- sapply(b,f)
```

We plot the results:

```r
plot(b,-ll,type='l',ylab=expression(-log(L)))
b.hat <- b[which.max(ll)]
abline(v=b.hat,lty=2)
```



The great similarity in the likelihood estimate to our first least-squares estimate is no accident. Why is this? Let y_t be the observed number of infectives at time t and Y_t be the model's prediction. Then the log likelihood is

$$\begin{aligned}\log \mathbb{P}\left[y_t|Y_t\right] &= \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_t-Y_t)^2}{2\sigma^2}\right)\right)\\ &= -\tfrac{1}{2}\log 2\pi\sigma^2 - \tfrac{1}{2}\frac{(y_t-Y_t)^2}{\sigma^2}\end{aligned}$$

and

$$\log\mathcal{L} = -\tfrac{1}{2}\left(\frac{1}{\sigma^2}\sum_t (y_t-Y_t)^2 + \log\left(\sigma^2\right) + \log\left(2\pi\right)\right)$$

So MLE and least-squares are equivalent if the errors are normal with constant variance!

Exercise 9. Suppose, alternatively, that the errors are log-normal with constant variance. Under what definition of SSE will least-squares and maximum likelihood give the same parameter estimates?

7 Modeling the noise

All this raises the question of what the best model for the errors really is. Of course, the answer will certainly depend on the nature of the data. The philosophy of likelihood encourages us to think about the question mechanistically. When the data, y_t, are the result of a sampling process, for example, we can think of them as binomial samples

$$y_t \sim \text{binomial}\left(Y_t, \frac{n}{N}\right)$$

where n is the sample size, N the population size, and Y_t is the true number of infections at time t. Alternatively, we might think of y_t as Poisson samples

$$y_t \sim \text{Poisson}(p\, Y_t)$$

where the parameter p reflects a combination of sampling efficiency and the detectability of infections. The latter leads to the following log-likelihood function

```
poisson.loglik <- function (params, data) {
  times <- data$biweek/26
  pred <- prediction(params,times)
  sum(dpois(x=data$measles,lambda=params["p"]*pred[-1],log=TRUE))
}
```

Let's see what the MLE parameters are for this model. We'll start by estimating just one parameter. Now, we must have $b > 0$. This is a *constraint* on the parameter. One way to enforce this constraint is by transforming the parameter so that it cannot ever be negative. We'll log-transform b.

```
dat <- subset(niamey,community=="A")
params <- c(X.0=20000,Y.0=1,gamma=365/13,b=NA,p=0.2)

## objective function (-log(L))
f <- function (log.b) {
  params[c("b")] <- exp(log.b) # un-transform 'b'
  -poisson.loglik(params,dat)
}
```

For something new, we'll use the `mle2` function from the bbmle package to maximize the likelihood. `mle2` will employ an iterative algorithm for maximizing the likelihood. To get started, it needs an initial guess.

```
require(bbmle)
guess <- list(log.b=log(0.01))

fit0 <- mle2(f,start=guess)
print(fit0)
```

```
## An object of class "mle2"
## Slot "call":
## mle2(minuslogl = f, start = guess, lower = -Inf, upper = Inf,
##     control = list())
##
## Slot "call.orig":
## mle2(minuslogl = f, start = guess)
##
## Slot "coef":
##     log.b
## -5.977186
##
## Slot "fullcoef":
##     log.b
## -5.977186
##
## Slot "vcov":
##              log.b
## log.b 2.565754e-06
##
## Slot "min":
## [1] 1963.317
##
## Slot "details":
## $par
##     log.b
## -5.977186
##
## $value
## [1] 1963.317
##
## $counts
## function gradient
##       46        9
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##        [,1]
## [1,] 389749
##
## $maxgrad
```

```
## [1] 1.421447
##
## $eratio
## [1] 1
##
##
## Slot "minuslogl":
## function (log.b) {
##   params[c("b")] <- exp(log.b) # un-transform 'b'
##   -poisson.loglik(params,dat)
## }
## <environment: 0x455be78>
##
## Slot "method":
## [1] "BFGS"
##
## Slot "data":
## list()
##
## Slot "formula":
## [1] ""
##
## Slot "optimizer":
## [1] "optim"
```

```
fit <-  mle2(f,start=as.list(coef(fit0)))
print(fit)
```

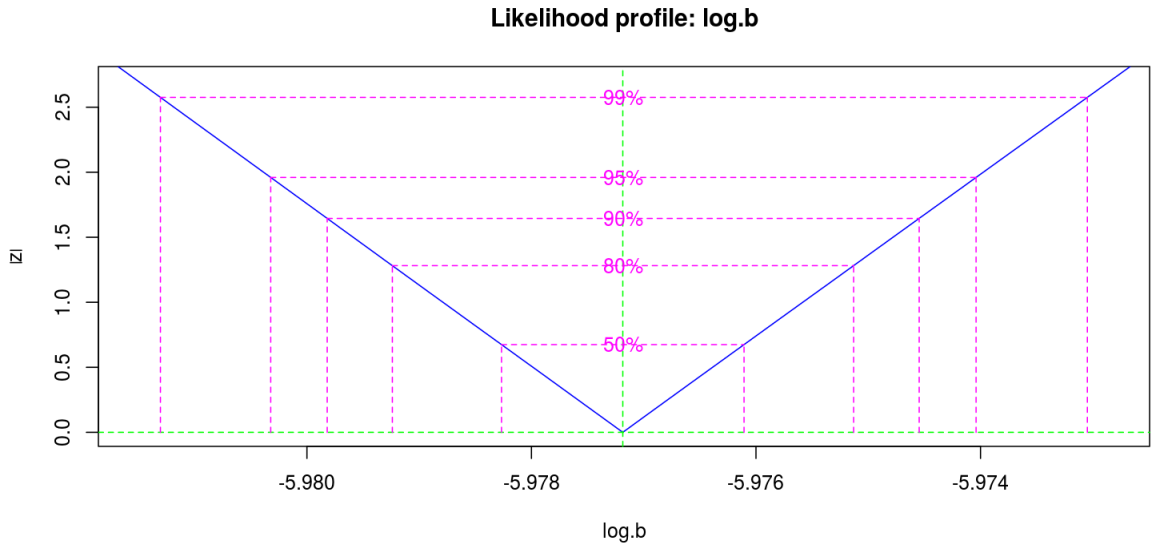
```
## An object of class "mle2"
## Slot "call":
## mle2(minuslogl = f, start = as.list(coef(fit0)), lower = -Inf,
##     upper = Inf, control = list())
##
## Slot "call.orig":
## mle2(minuslogl = f, start = as.list(coef(fit0)))
##
## Slot "coef":
##     log.b
## -5.977186
##
## Slot "fullcoef":
##     log.b
## -5.977186
##
## Slot "vcov":
```

```
##              log.b
## log.b 2.565754e-06
##
## Slot "min":
## [1] 1963.317
##
## Slot "details":
## $par
##     log.b
## -5.977186
##
## $value
## [1] 1963.317
##
## $counts
## function gradient
##       12        1
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##        [,1]
## [1,] 389749
##
## $maxgrad
## [1] 1.421508
##
## $eratio
## [1] 1
##
##
## Slot "minuslogl":
## function (log.b) {
##   params[c("b")] <- exp(log.b) # un-transform 'b'
##   -poisson.loglik(params,dat)
## }
## <environment: 0x66ee3c0>
##
## Slot "method":
## [1] "BFGS"
##
## Slot "data":
```

```
## list()
##
## Slot "formula":
## [1] ""
##
## Slot "optimizer":
## [1] "optim"
```

We can get an idea about the uncertainty and in particular obtain confidence intervals using the *profile likelihood*. To profile over a parameter, we fix the value of that parameter at each of several values, then maximize the likelihood over the remaining unknown parameters. In bbolme, this is quite easy to obtain.

```
prof.b <- profile(fit)
plot(prof.b)
```



Now let's try to estimate both b and the reporting probability p. Since we have constraints on p ($0 \leq p \leq 1$), we'll transform it as well. For this, the *logit* function and its inverse are useful:

$$\text{logit}(p) = \log \frac{p}{1-p} \qquad \text{ilogit}(x) = \frac{1}{1+\exp(-x)}$$

```
dat <- subset(niamey,community=="A")
params <- c(X.0=20000,Y.0=1,gamma=365/13,b=NA,p=NA)

logit <- function (p) log(p/(1-p))      # the logit transform
ilogit <- function (x) 1/(1+exp(-x))    # inverse logit
```

```r
f <- function (log.b, logit.p) {
  par <- params
  par[c("b","p")] <- c(exp(log.b),ilogit(logit.p))
  -poisson.loglik(par,dat)
}

guess <- list(log.b=log(0.005),logit.p=logit(0.2))
fit0 <- mle2(f,start=guess); fit0


## An object of class "mle2"
## Slot "call":
## mle2(minuslogl = f, start = guess, lower = -Inf, upper = Inf,
##     control = list())
##
## Slot "call.orig":
## mle2(minuslogl = f, start = guess)
##
## Slot "coef":
##      log.b    logit.p
## -5.9918688 -0.1470357
##
## Slot "fullcoef":
##      log.b    logit.p
## -5.9918688 -0.1470357
##
## Slot "vcov":
##                  log.b       logit.p
## log.b    2.882471e-06 -9.272553e-06
## logit.p -9.272553e-06  6.151853e-04
##
## Slot "min":
## [1] 394.1967
##
## Slot "details":
## $par
##      log.b    logit.p
## -5.9918688 -0.1470357
##
## $value
## [1] 394.1967
##
## $counts
## function gradient
##       51       10
##
```

```
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##             [,1]     [,2]
## [1,] 364603.198 5495.584
## [2,]   5495.584 1708.360
##
## $maxgrad
## [1] 0.04106786
##
## $eratio
## [1] 0.00445631
##
##
## Slot "minuslogl":
## function (log.b, logit.p) {
##   par <- params
##   par[c("b","p")] <- c(exp(log.b),ilogit(logit.p))
##   -poisson.loglik(par,dat)
## }
## <environment: 0x60384f8>
##
## Slot "method":
## [1] "BFGS"
##
## Slot "data":
## list()
##
## Slot "formula":
## [1] ""
##
## Slot "optimizer":
## [1] "optim"
```

```
fit <-  mle2(f,start=as.list(coef(fit0))); print(fit)
```

```
## An object of class "mle2"
## Slot "call":
## mle2(minuslogl = f, start = as.list(coef(fit0)), lower = -Inf,
##     upper = Inf, control = list())
##
```

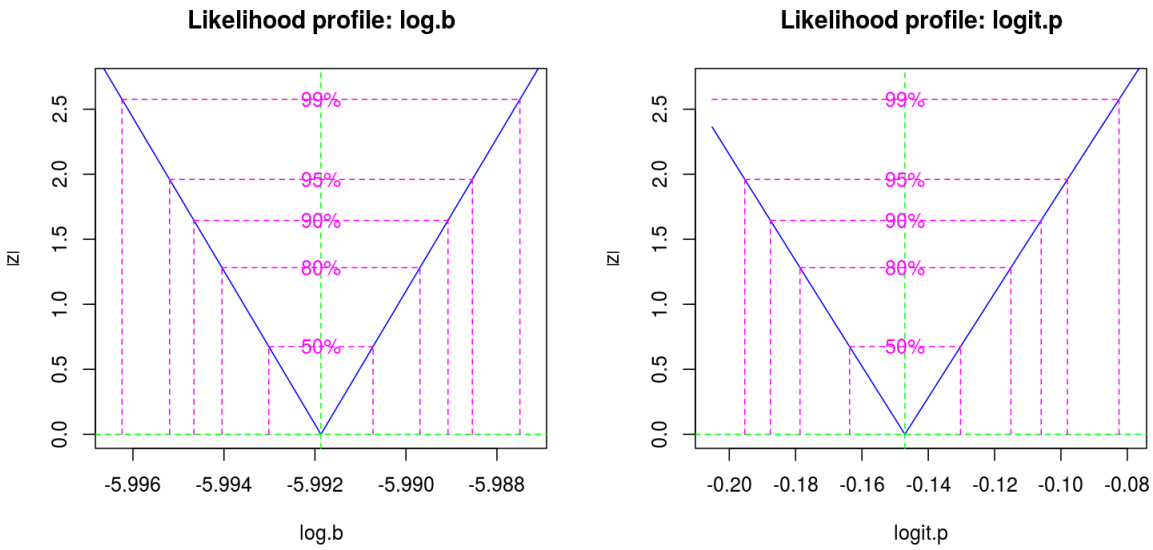
```
## Slot "call.orig":
## mle2(minuslogl = f, start = as.list(coef(fit0)))
##
## Slot "coef":
##      log.b    logit.p
## -5.9918687 -0.1470357
##
## Slot "fullcoef":
##      log.b    logit.p
## -5.9918687 -0.1470357
##
## Slot "vcov":
##                  log.b       logit.p
## log.b    2.882471e-06 -9.272548e-06
## logit.p -9.272548e-06  6.151852e-04
##
## Slot "min":
## [1] 394.1967
##
## Slot "details":
## $par
##      log.b    logit.p
## -5.9918687 -0.1470357
##
## $value
## [1] 394.1967
##
## $counts
## function gradient
##       11        1
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##            [,1]     [,2]
## [1,] 364603.158 5495.581
## [2,]   5495.581 1708.360
##
## $maxgrad
## [1] 0.01437765
##
## $eratio
```

```
## [1] 0.004456311
##
##
## Slot "minuslogl":
## function (log.b, logit.p) {
##   par <- params
##   par[c("b","p")] <- c(exp(log.b),ilogit(logit.p))
##   -poisson.loglik(par,dat)
## }
## <environment: 0x5519cd8>
##
## Slot "method":
## [1] "BFGS"
##
## Slot "data":
## list()
##
## Slot "formula":
## [1] ""
##
## Slot "optimizer":
## [1] "optim"
```

```
## now untransform the parameters:
mle <- with(
            as.list(coef(fit)),
            c(
              b=exp(log.b),
              p=ilogit(logit.p)
              )
            )
mle
```

```
##          b          p
## 0.00249899 0.46330717
```

```
prof2 <- profile(fit)
plot(prof2)
```



We can also get confidence intervals:

```
ci <- confint(prof2)
ci
```

```
##               2.5 %      97.5 %
## log.b   -5.9951952 -5.98854089
## logit.p -0.1953059 -0.09803465
```

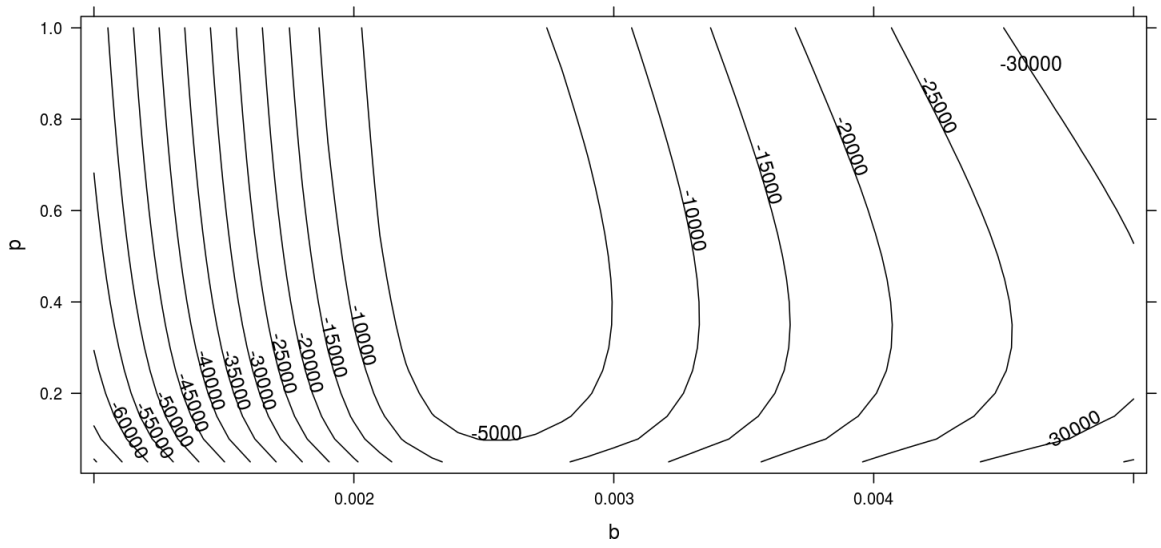
```
ci[1,] <- exp(ci[1,])
ci[2,] <- ilogit(ci[2,])
rownames(ci) <- c("b","p")
ci
```

```
##          2.5 %     97.5 %
## b 0.002490691 0.00250732
## p 0.451328148 0.47551095
```

Let's make a contour plot to visualize the likelihood surface.

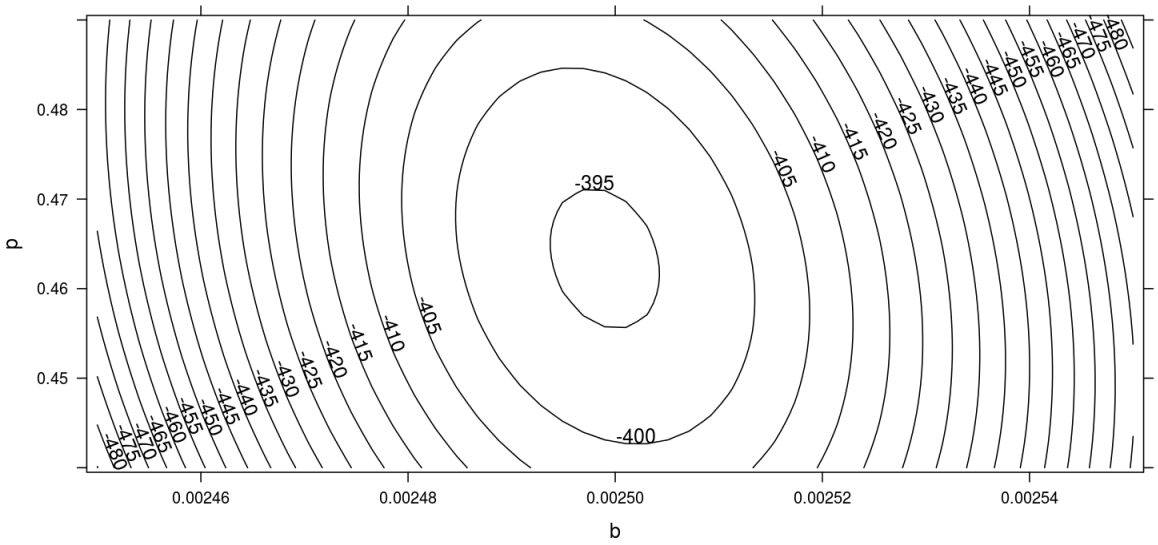
```
dat <- subset(niamey,community=="A")

## this time the objective function has to
## take a vector argument
f <- function (pars) {
  par <- params
  par[c("b","p")] <- as.numeric(pars)
```

```
  poisson.loglik(par,dat)
}

b <- seq(from=0.001,to=0.005,by=0.0001)
p <- seq(0,1,by=0.05)
grid <- expand.grid(b=b,p=p)
grid$loglik <- apply(grid,1,f)
grid <- subset(grid,is.finite(loglik))
require(lattice)
contourplot(loglik~b+p,data=grid,cuts=20)
```



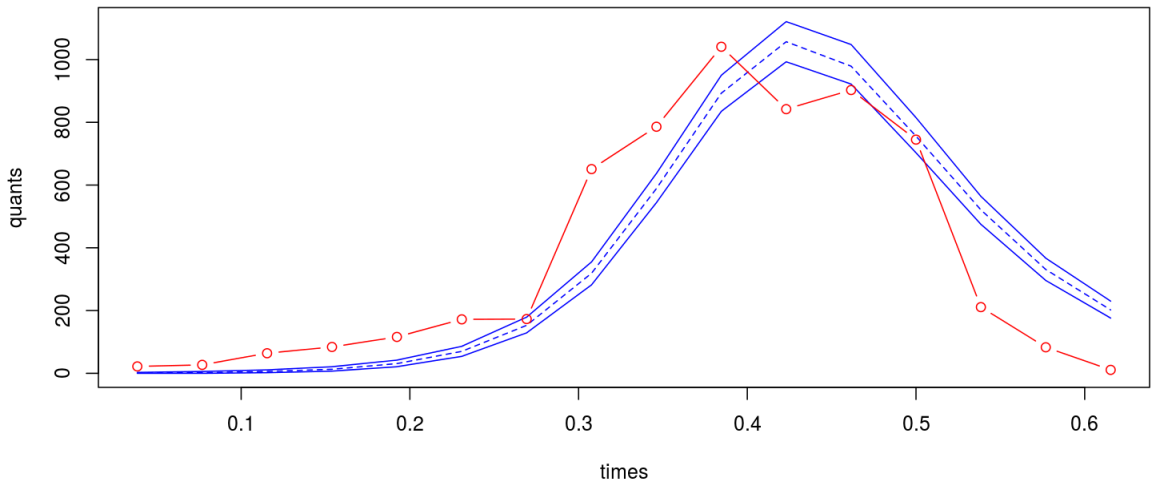
The scale over which the log likelihood is varying is clearly huge relative to what is meaningful. Let's focus in on the region around the MLE.

```
b <- seq(from=0.00245,to=0.00255,length=50)
p <- seq(0.44,0.49,length=50)
grid <- expand.grid(b=b,p=p)
grid$loglik <- apply(grid,1,f)
grid <- subset(grid,is.finite(loglik))
require(lattice)
contourplot(loglik~b+p,data=grid,cuts=20)
```



Let's look at the model's predictions at the MLE. The model is a probability distribution, so we should look at a number of simulations. An important question is: are the data a plausible sample from the predicted probability distribution?

```
params[c("b","p")] <- mle
times <- c(dat$biweek/26)
model.pred <- prediction(params,times)

nsim <- 1000
simdat <- replicate(
                    n=nsim,
                    rpois(n=length(model.pred),
                          lambda=params["p"]*model.pred)
                    )
quants <- t(apply(simdat,1,quantile,probs=c(0.025,0.5,0.975)))
matplot(times,quants,col="blue",lty=c(1,2,1),type='l')
points(measles~times,data=dat,type='b',col='red')
```



Clearly the model is not doing a very good job of capturing the pattern in the data. It appears that we will need an error model that has the potential for more variability than does the Poisson. Recall that, under the Poisson assumption, the variance of the error is equal to the mean. The negative binomial distribution is such a distribution. Let's explore the alternative assumption that y_t is negative-binomially distributed with mean $p\,Y_t$, as before, but larger variance, $p\,Y_t\,(1+\theta\,p\,Y_t)$, i.e.,

$$y_t \;\sim\; \text{Negbin}\left(\text{mu} = p\,Y_t,\; \text{size} = \frac{1}{\theta}\right)$$

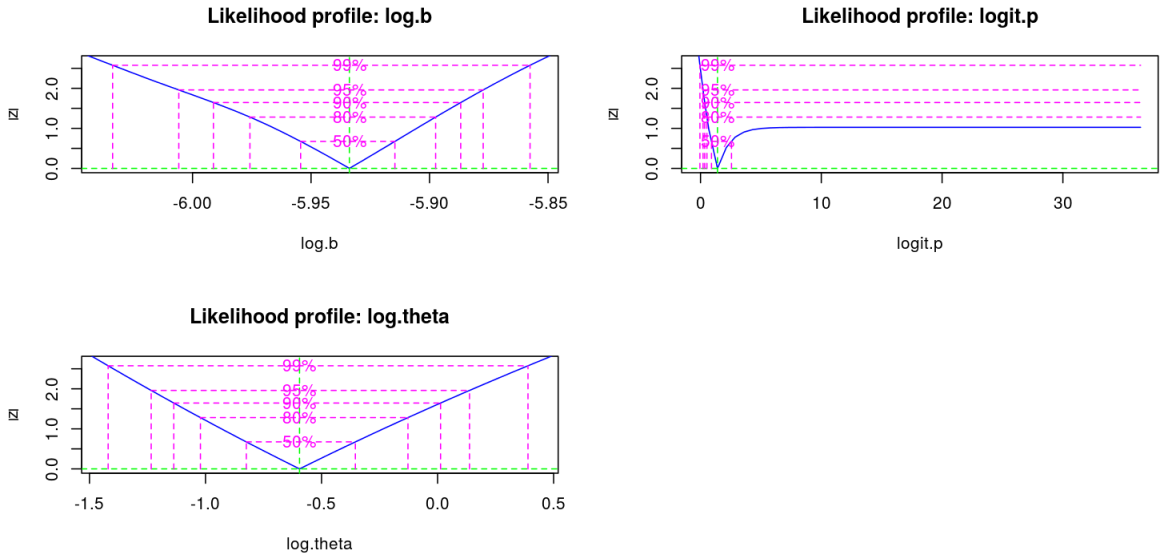
```
loglik <- function (params, data) {
  times <- data$biweek/26
  pred <- prediction(params,times)
  sum(dnbinom(x=data$measles,
              mu=params["p"]*pred[-1],size=1/params["theta"],
              log=TRUE))
}

f <- function (log.b, logit.p, log.theta) {
  par <- params
  par[c("b","p","theta")] <- c(exp(log.b),
                               ilogit(logit.p),
                               exp(log.theta))
  -loglik(par,dat)
}

guess <- list(log.b=log(params["b"]),
              logit.p=logit(params["p"]),
              log.theta=log(1))
```

```r
fit0 <- mle2(f,start=guess)
fit <-  mle2(f,start=as.list(coef(fit0)))
print(fit)
```

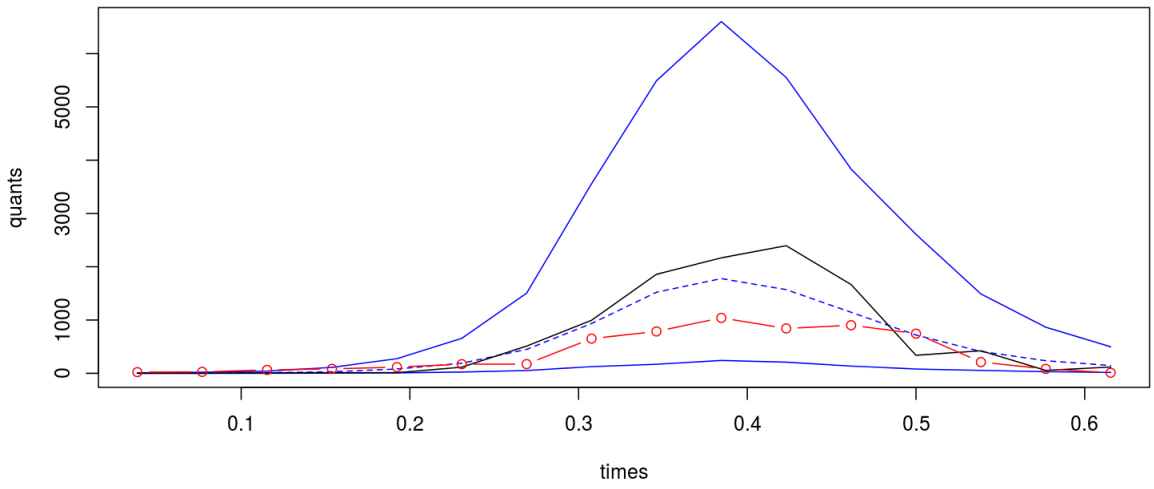
```
## An object of class "mle2"
## Slot "call":
## mle2(minuslogl = f, start = as.list(coef(fit0)), lower = -Inf,
##     upper = Inf, control = list())
##
## Slot "call.orig":
## mle2(minuslogl = f, start = as.list(coef(fit0)))
##
## Slot "coef":
##     log.b   logit.p log.theta
## -5.933817  1.407372 -0.594814
##
## Slot "fullcoef":
##     log.b   logit.p log.theta
## -5.933817  1.407372 -0.594814
##
## Slot "vcov":
##                   log.b      logit.p    log.theta
## log.b      0.0008464986 -0.005478065 0.0004644563
## logit.p   -0.0054780649  1.036557802 0.0340473405
## log.theta  0.0004644563  0.034047340 0.1211614730
##
## Slot "min":
## [1] 102.4639
##
## Slot "details":
## $par
##     log.b   logit.p log.theta
## -5.933817  1.407372 -0.594814
##
## $value
## [1] 102.4639
##
## $counts
## function gradient
##       12        1
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
##
## $hessian
##             [,1]       [,2]       [,3]
## [1,] 1228.367361  6.7083381 -6.5938786
## [2,]    6.708338  1.0103546 -0.3096332
## [3,]   -6.593879 -0.3096332  8.3657348
##
## $maxgrad
## [1] 0.005287511
##
## $eratio
## [1] 0.0007843522
##
##
## Slot "minuslogl":
## function (log.b, logit.p, log.theta) {
##   par <- params
##   par[c("b","p","theta")] <- c(exp(log.b),
##                                ilogit(logit.p),
##                                exp(log.theta))
##   -loglik(par,dat)
## }
## <environment: 0x59e4f20>
##
## Slot "method":
## [1] "BFGS"
##
## Slot "data":
## list()
##
## Slot "formula":
## [1] ""
##
## Slot "optimizer":
## [1] "optim"
```

```
prof3 <- profile(fit)
plot(prof3)
```




```
mle <- with(
            as.list(coef(fit)),
            c(
              b=exp(log.b),
              p=ilogit(logit.p),
              theta=exp(log.theta)
              )
            )

params[c("b","p","theta")] <- mle
times <- c(dat$biweek/26)
model.pred <- prediction(params,times)

nsim <- 1000
simdat <- replicate(
                    n=nsim,
                    rnbinom(n=length(model.pred),
                            mu=params["p"]*model.pred,
                            size=1/params["theta"])
                    )
quants <- t(apply(simdat,1,quantile,probs=c(0.025,0.5,0.975)))
matplot(times,quants,col="blue",lty=c(1,2,1),type='l')
lines(times,simdat[,1],col='black')
points(measles~times,data=dat,type='b',col='red')
```



What does this plot tell us? Essentially, the deterministic SIR model, as we've written it, cannot capture the shape of the epidemic. In order to fit the data, the optimization algorithm has expanded the error variance, to the point of absurdity. The typical model realization (in black) does not much resemble the data.

Exercise 10. Revisit the other communities of Niamey and/or the British boarding school influenza data using the Poisson model and bbmle.

***Exercise 11.** Try to estimate p, b, and X_0 simultaneously.

***Exercise 12.** Reformulate the problem using the binomial error model. Modify the parameter estimation codes appropriately, estimate the parameters, and comment on the results.

***Exercise 13.** Reformulate the problem using a normal error model in which the variance is proportional to the mean:

$$y_t \sim \text{normal}\left(p\,Y_t, \sigma\,\sqrt{Y_t}\right).$$

Modify the parameter estimation codes appropriately, estimate the parameters (including both p and σ), and comment on the results.

***Exercise 14.** We've been treating the Niamey data as if they were direct—though inaccurate—measurements of the prevalence. Actually, these are incidence data: they are measures of unique infections. It would be more appropriate to model these data by adding another equation

$$\frac{dC}{dt} = \frac{\beta\,X\,Y}{N}$$

to accumulate new infections and assuming the data are distributed according to, for example,

$$y_t \sim \text{Poisson}\left(p\,(C_t - C_{t-1})\right).$$

Modify the codes above to reflect these more appropriate assumptions, estimate the parameters, and comment on the results.