Basic SIR fitting

Ben Bolker, David Earn, Dora Rosati

October 1, 2014

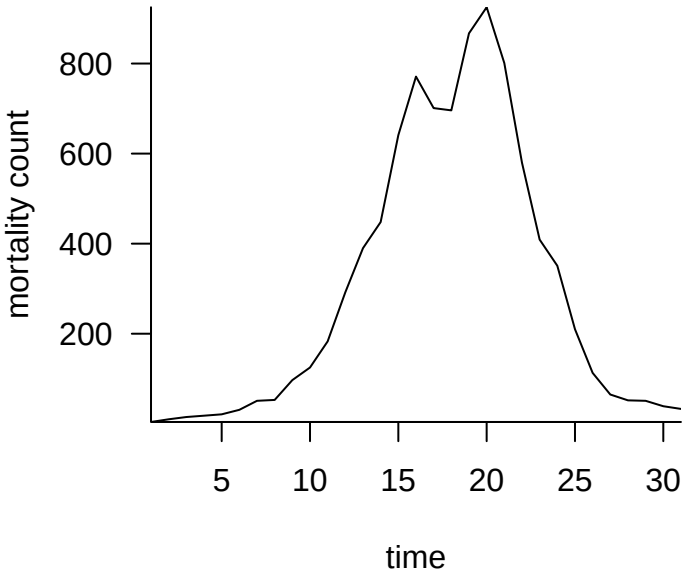
This has been done a million times, but let's try to do it in a reasonably systematic way that could be used in a pedagogical paper.

```
library("fitsir")
library("bbmle") ## need this for now, for coef()
library("plyr")
library("reshape2")
library("ggplot2"); theme_set(theme_bw())
```

The current version of `fitsir` assumes that time and prevalence are stored as columns `tvec` and `count` within a data frame. Since the `bombay` data set instead has `week` (week of epidemic) and `mort` (mortality), we'll rename it for convenience. (We will for now resolutely ignore issues about fitting weekly mortality counts as prevalences ...)

```
bombay2 <- setNames(bombay,c("tvec","count"))
```

```
plot(count~tvec,data=bombay2,
     type="l",xaxs="i",yaxs="i",
     xlab="time",ylab="mortality count")
```



1 Fit the model to the data

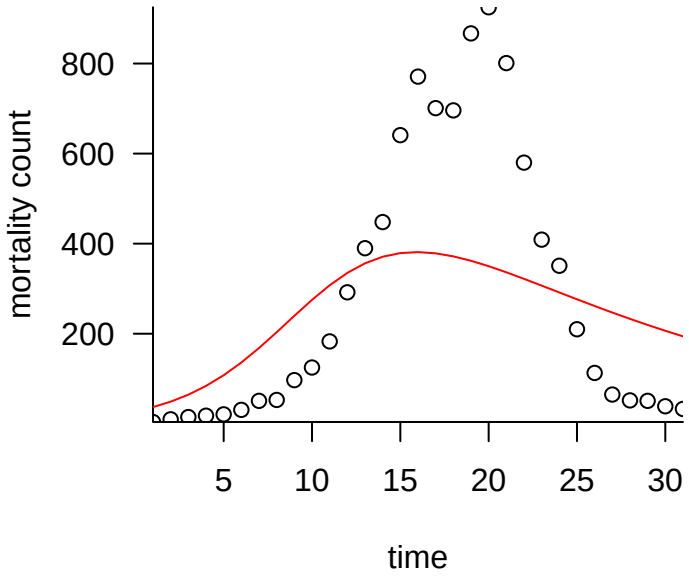
Basic fit:

```
m1 <- fitsir(data=bombay2)
```

```
summarize.pars(coef(m1))

##          R0           r      infper          i0
##  5.13774293  0.28629956 14.45249473  0.05235677
```

Seemingly reasonable answers, but ...

```
ss <- with(bombay2,SIR.detsim(tvec,trans.pars(coef(m1))))
plot(count~tvec,data=bombay2,
     xaxs="i",yaxs="i",
     xlab="time",ylab="mortality count")
lines(bombay2$tvec,ss,col=2)
```



2 Troubleshooting

We're obviously not getting a good answer here. When this happens there are a variety of possibilities.

- optimizer getting stuck
- a small number of local optima
- a large number of local optima, on many different scales (fractal-like or rugged surface)
- a large number of local optima, all similar in scale/height ("fakir's bed" geometry)

Some solutions:

- center/scale parameters and/or reparameterize the model to remove correlation and equalize scales of variation in different parameters
- try to come up with a rule for finding better starting values ("self-starting" fits)

- use a better/more robust local optimizer
- use lots of starting values, randomly or regularly distributed
- use a stochastic global optimizer

```
confint(m1,method="quad")

## Warning:  NaNs produced

##                2.5 %     97.5 %
## log.beta  -1.453650 -0.6148568
## log.gamma -2.748828 -2.5929057
## log.N            NaN        NaN
## logit.i   -4.049810 -1.7419839
```

Suggests *some* sort of unidentifiability ...

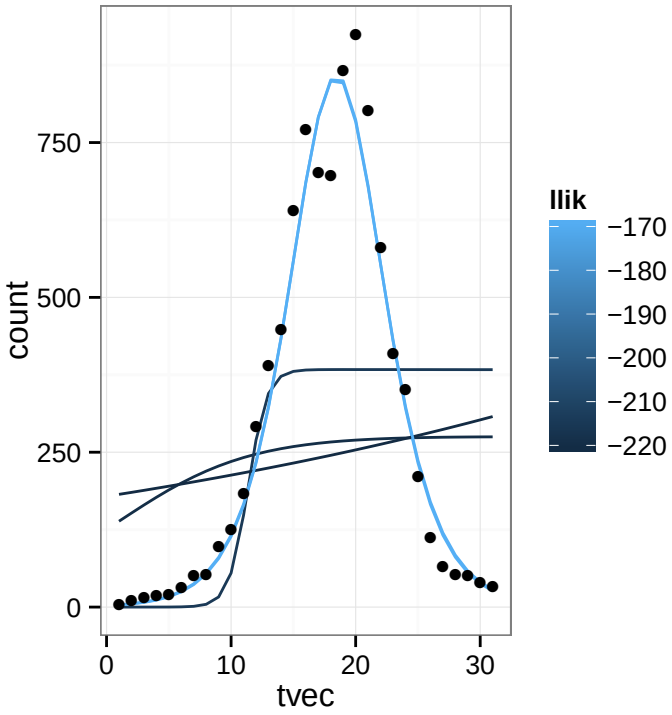
What if we try a bunch of starting values?

A crude Latin-hypercube-like strategy: pick evenly spaced values on sensible log scales, then permute to get random (but even) coverage of the space.

```
qlhcfun <- function(n=5,seed=NULL) {
    require("plyr")
    if (!is.null(seed)) set.seed(seed)
    R0vec <- 1+10^seq(-1,1.5,length=n)
    infpervec <- sample(10^seq(-1,2,length=n))
    Nvec <- sample(10^seq(2,5,length=n))
    i0vec <- sample(10^seq(-3,-1,length=n))
    startlist <- alply(cbind(R0=R0vec,infper=infpervec,N=Nvec,i0=i0vec),1,
                       function(x) {
                           with(as.list(x), {
                               beta <- R0/infper
                               gamma <- 1/infper
                               c(log.beta=log(beta),log.gamma=log(gamma),
                                 log.N=log(N),logit.i=qlogis(i0))
                           })
                       })
    return(startlist)
}
startlist <- qlhcfun(n=5,seed=101)
```

```
fitlist <- llply(startlist,fitsir,data=bombay2,
      method="Nelder-Mead",control=list(maxit=1e5))
```

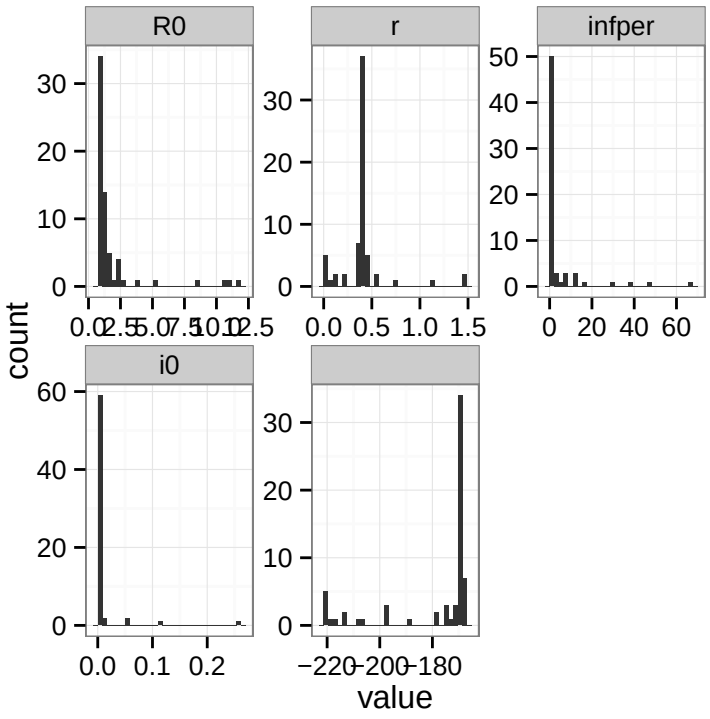
```
## extract log-likelihoods
likframe <- data.frame(.id=1:5,llik=unlist(llply(fitlist,logLik)))
## compute trajectories
gettraj <- function(x,tvec=bombay2$tvec) {
    data.frame(tvec=tvec,
               count=SIR.detsim(tvec,trans.pars(coef(x))))
}
fittraj <- ldply(fitlist,gettraj)
fittraj <- merge(fittraj,likframe)
## plot together
ggplot(fittraj,aes(tvec,count,colour=llik,group=.id))+geom_line()+
    geom_point(data=bombay2,colour="black",aes(group=NA))
```



Now try a much larger sample:

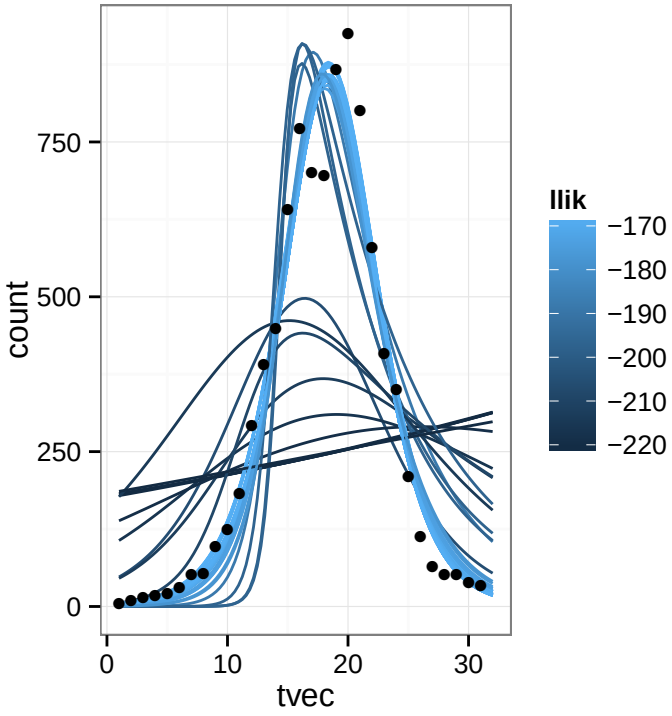
```
startlist100 <- qlhcfun(n=100,seed=101)
fitlist100 <- llply(startlist100,
                    function(x) {
                        r <- try(fitsir(start=x,data=bombay2),silent=TRUE)
                        if (is(r,"try-error")) NULL else r
                    })
```

```
testOK  <- function(x,max.R0=100,max.r=1000,max.infper=400) {
    if (is.null(x)) return(FALSE)
    ss <- summarize.pars(coef(x))
    return(ss["R0"]<max.R0 & ss["r"]<max.r & ss["infper"] < max.infper)
}
fitlist100.OK <- fitlist100[sapply(fitlist100,testOK)]
length(fitlist100.OK)

## [1] 65

fittab <- laply(fitlist100.OK,function(x) c(summarize.pars(coef(x)),logLik(x)))
ggplot(melt(fittab),aes(x=value))+geom_histogram()+facet_wrap(~Var2,scale="free")

## Warning:  position_stack requires constant width:  output may be
incorrect
```



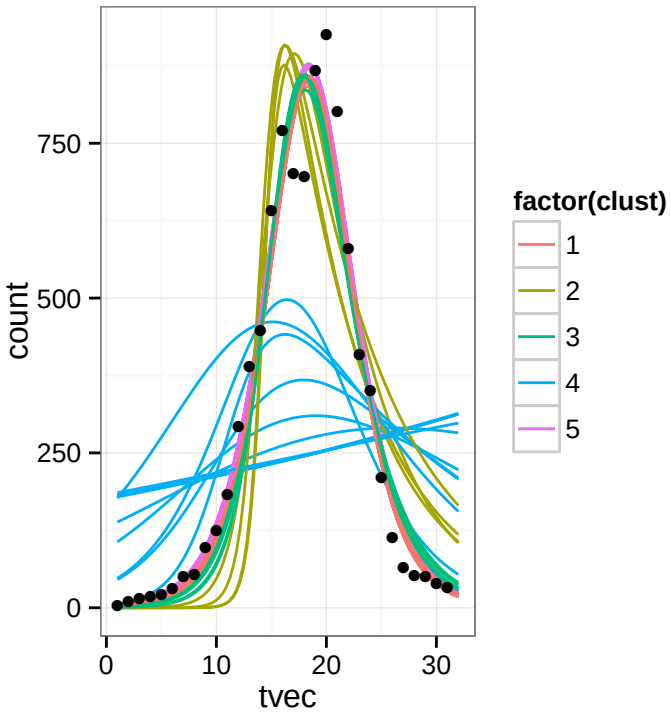

```
likframe100 <- setNames(ldply(fitlist100.OK,logLik),c(".id","llik"))
fittraj100 <- ldply(fitlist100.OK,gettraj,tvec=seq(1,32,length=101))
fitmat100 <- acast(fittraj100,tvec~.id,value.var="count")
fittraj100 <- merge(fittraj100,likframe100)
## plot together
```

```
ggplot(fittraj100,aes(tvec,count,colour=llik,group=.id))+geom_line()+
    geom_point(data=bombay2,colour="black",aes(group=NA))
```



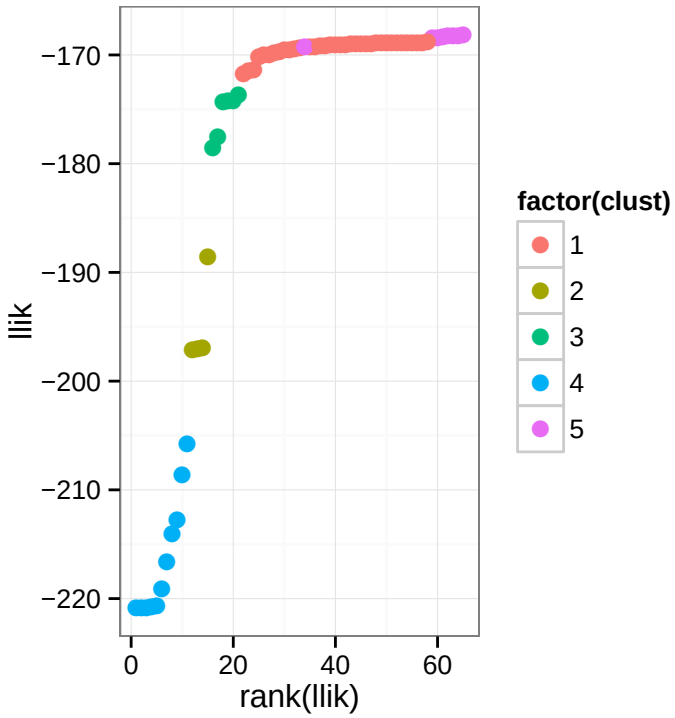
We can identify clusters ...

```
clust <- kmeans(t(fitmat100),5)
cframe <- data.frame(.id=names(clust$cluster),clust=clust$cluster)
fittraj100B <- transform(fittraj100,llikcat=cut_number(llik,5))
fittraj100B <- merge(fittraj100B,cframe)
ggplot(fittraj100B,aes(tvec,count,colour=factor(clust),group=.id))+geom_line()+
    geom_point(data=bombay2,colour="black",aes(group=NA))
```



Check out clustering on log-likelihood cumulative distribution curve:

```
dd2 <- merge(cframe,likframe100)
(g1 <- ggplot(dd2,aes(rank(llik),llik,colour=factor(clust)))+geom_point(size=3))
```



I'm not 100% sure (yet) what this tells us. The clusters aren't so well separated that I necessarily believe that they are distinct modes.